NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

USING DISCRETE EVENT SIMULATION TO ASSESS OBSTACLE LOCATION ACCURACY IN THE REMUS UNMANNED UNDERWATER VEHICLE

by

Timothy E. Allen

June 2004

Thesis Advisor:	Arnold H. Buss
Second Reader:	Susan M. Sanchez

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Using Discrete Event Simulation to Assess Obstacle Location Accuracy in the REMUS Unmanned Underwater Vehicle			**5. FUNDING NUMBERS**
6. AUTHOR(S) Timothy E. Allen			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Navy personnel use the REMUS unmanned underwater vehicle to search for submerged objects. Navigation inaccuracies lead to errors in predicting the location of objects and thus result in increased search times for EOD teams searching for the object post mission. This thesis explores contributions to navigation inaccuracy using Discrete Event Simulation to model the vehicle's navigation system and operational performance. The DES generates data used, in turn, to build three models. First, the probability of detection is modeled by a logit regression. Second, given that detection has occurred, the mean location offset is modeled by a linear regression. Third, the distribution of errors is shown to follow an exponential distribution. These three models enable operators to explore the impact of various inputs prior to programming the vehicle, thus allowing them to choose the best combination of vehicle parameters that minimize the offset error between the reported and actual locations.

14. SUBJECT TERMS Mine, Minefield, Simulation, Discrete Event Simulation, Simkit, Navigation			**15. NUMBER OF PAGES** 149
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

USING DISCRETE EVENT SIMULATION TO ASSESS OBSTACLE LOCATION ACCURACY IN THE REMUS UNMANNED UNDERWATER VEHICLE

Timothy E. Allen
Lieutenant, United States Navy
B.S., University of Idaho, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS ANALYSIS

from the

NAVAL POSTGRADUATE SCHOOL
June 2004

Author: Timothy E. Allen

Approved by: Arnold H. Buss
Thesis Advisor

Susan M. Sanchez
Second Reader

James Eagle
Chairman, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Navy personnel use the REMUS unmanned underwater vehicle to search for submerged objects. Navigation inaccuracies lead to errors in predicting the location of objects and thus result in increased search times for EOD teams searching for the object post mission. This thesis explores contributions to navigation inaccuracy using Discrete Event Simulation to model the vehicle's navigation system and operational performance. The DES generates data used, in turn, to build three models. First, the probability of detection is modeled by a logit regression. Second, given that detection has occurred, the mean location offset is modeled by a linear regression. Third, the distribution of errors is shown to follow an exponential distribution. These three models enable operators to explore the impact of various inputs prior to programming the vehicle, thus allowing them to choose the best combination of vehicle parameters that minimize the offset error between the reported and actual locations.

THIS PAGE INTENTIONALLY LEFT BLANK

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made within the time available to ensure the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Professors Arnold Buss and Susan Sanchez. Their assistance was instrumental in making this thesis a success. I would like to thank Doug Horner for his insights and assistance in understanding the vehicle's behavior. Also, to the many other professors and classmates willing to listen as I bounced ideas off of them, thank you. And most importantly, I owe a debt of gratitude to my loving wife, Jeannette, and my three wonderful children, Stephanie, Melissa and Danny. Without their support, patience and understanding, none of this work would have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Naval vessels have always been susceptible to mine warfare. Whether the threat is real or simply perceived, the end result is the same: mine warfare disrupts the ability to project and maintain forces away from home waters. The United States Navy has placed a high priority on developing technology and tactics designed to counter the threat of mine warfare. One of the most promising is the REMUS unmanned underwater vehicle. The REMUS shows potential as a minefield survey tool, but an analysis of the effects of environmental conditions on its navigational accuracy, and thus its ability to effectively pinpoint the location of a submerged object, has not been adequately documented.

This thesis undertakes such an analysis using Discrete Event Simulation (DES) to model the navigation system. DES is tool specifically suited for this type of analysis. Large amounts of data are not available from actual missions, and attempting to conduct enough missions to collect data is not feasible because of the amount of time required. There are simply too many variables to consider. DES allows for data collection in the absence of real data by building scenarios that mimic true operating conditions and using the generated data to analyze mission performance. In so far as the environmental conditions have been accurately implemented, the output provides insight into combinations of operating conditions to exploit, or avoid.

The DES produced for this thesis uses the JAVA based Simkit package to simulate the navigation system in the REMUS. The model considers factors affecting accuracy, such as compass error, the effect of current, transducer drop error, transducer positioning effects and ping interval. Mines can be placed at specific locations or generated randomly. The vehicle can be programmed to proceed to as many waypoints as desired, thus allowing the vehicle to conduct a search pattern through an area of interest. The size and layout of the search area is limited only by factors which also limit the real vehicle's performance, such as sound propagation in water.

Three types of vehicles are considered in this thesis. First, a simple vehicle that navigates by Dead Reckoning is analyzed. The vehicle is simulated to move from one point to another and one mine is placed in its path. Second, a more complex vehicle that

navigates using Long-Baseline (LBL) is analyzed. The vehicle is again simulated to move from one point to another with a single mine in its path. Third, the vehicle is simulated to move through an area of interest in a sweeping, “mow the lawn”, pattern which is populated by ten mines, each of which is randomly positioned. This third vehicle most accurately reflects the current technique for operating the real vehicle.

The data from the first two versions of the simulation are used to generate insight into how factors influence the vehicle’s prediction performance. Data from the last version of the simulation are used to build three analytic models that the operator can use to improve performance. First, a logit regression model predicts the probability of detection of a mine given a combination of input parameters. This model allows operators to explore combinations of inputs in an attempt to maximize prediction power. Second, a linear regression model predicts the mine location error, given that detection has occurred. This model allows operators to explore inputs in an effort to reduce the prediction error. Finally, the distribution of errors from the second model is shown to follow an exponential distribution. The results of the third model serve to greatly simplify probability calculations by operators in the field.

Analysis of the three versions of the simulation provides useful insights into vehicle performance. Current direction and speed influence accuracy in several ways. Higher current speeds cause the vehicle to be pushed off track, and current directions that are perpendicular to the vehicle’s track degrade detection probability more than aiding or opposing currents. Currents that tend to push the vehicle from behind result in the best performance, both in terms of detection probability and location error. Current also influences accuracy by causing the transducers to be displaced from their pre-programmed location. Shorter ping intervals also improve performance mainly because position fixes are calculated more often. There is a trade-off, however, because a short ping interval reduces the effective range at which the vehicle can operate from the transducers and thus shrinks the effective search area.

This thesis provides an exploratory model for the REMUS vehicle’s navigation system that an operator can use to improve performance while searching an area of interest for mines or other objects. However, all results should be viewed as just what

they are - insights. Simplifications are made in the model's implementation of complex hydrodynamic effects. Every effort has been made to produce a model that accurately reflects the vehicle's true performance, but users are cautioned that the output of the model should not be considered to be the only answer to the problem.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MINE WARFARE

Naval vessels have always been susceptible to mine warfare. Whether the threat is real or simply perceived, the end result is the same: mine warfare disrupts the ability to project and maintain forces away from home waters. The technology involved in construction and employment of mines has not changed appreciably in the past few decades. However, the United States Navy has placed a high priority on developing technology and tactics designed to counter the threat of mine warfare. Three US warships, USS SAMUEL B ROBERTS, USS PRINCETON and USS TRIPOLI, have been involved in encounters with mines since 1988 in the Persian Gulf.

Mine warfare is affordable and available to any country willing to invest the time required to lay the mine-field. In March 2003, during the second Gulf War, Navy ships intercepted three tug boats in the vicinity of the Iraqi port of Umm Qasar loaded with over 130 mines intended for the harbor inner reaches [Eisman, 2003]. This incident underscores the relative ease with which a country could employ this rather primitive, yet effective, tactic.

Mine warfare can be traced back in history as early as the American Revolution. David Bushnell invented a simple contact mine made from a watertight wooden keg filled with explosive powder. An illustration of the Bushnell mine is shown in Figure 1.

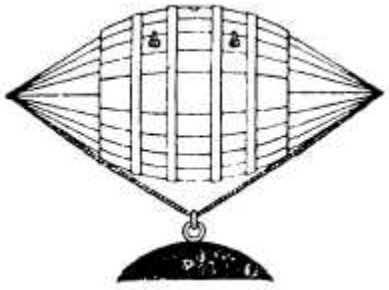


Figure 1. Bushnell Mine (From Mineman, Vol I, Ch I, 1994)

Mine technology has advanced substantially since Bushnell's early attempts. Mines may be laid in surf zones to thwart landing craft or in deep sea reaches to combat deep draft, capital ships. Some mines may be as crude as Bushnell's early design and

activate primarily by contact. Others are more complex and are activated by influence, acoustics or other more advanced mechanisms. Figure 2 shows the variations presently in use. Mineman [Vol I, Ch I, 1994] lists the following advantages that mines have over other conventional weapons:

- Mines lie in wait for the enemy with no reasonable threat of counter-detection
- Mines may win a conflict passively by causing the enemy to alter tactics
- Mines may force ships to travel longer, less reliable routes to deliver troops and materials
- Mines are cost effective due to their relatively primitive technology

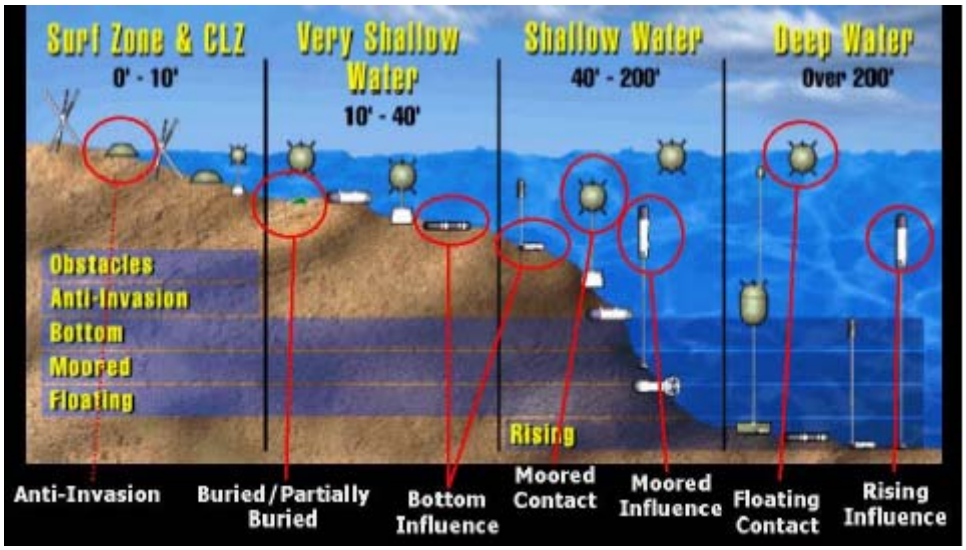


Figure 2. Mine Types Used in Various Situations (From Nawara)

Existing tools for mine detection and clearing include mine sweeping ships, trained animals, Explosive Ordnance Disposal (EOD) teams, helicopter squadrons and Unmanned Underwater Vehicles (UUV). Of these, the UUV is perhaps the most exciting and promising. Several variations of the UUV are in various stages of use and testing. The Naval Postgraduate School (NPS) in Monterey, CA currently possesses two capable machines: the ARIES and the Remote Environmental Measuring UnitS (REMUS). This thesis focuses on the REMUS. REMUS and vehicle will hereafter be used interchangeably.

B. REMUS BASICS

The REMUS is built by Hydroid Technologies and is designed to collect hydrographic data in relatively shallow water. The overall system is comprised of the vehicle, various auxiliary equipment necessary to support its mission and software programs designed to conduct pre-mission planning and post-mission data analysis. During a mission the vehicle collects side-scan images that can be viewed, post-mission, and used to identify and locate objects (e.g. mines, obstacles) on the ocean floor. Other data are collected and saved that are important in assessing the accuracy of the ostensible location of the object. Theoretically, returning to the aforementioned object is an easy task for the EOD teams because the vehicle always "knows where it is," but in reality the task is made more difficult due to inaccuracies in the navigation system that result in errors in the reported location of the objects.

The vehicle is 7.5 inches in diameter, 40 inches in length and weighs about 70 lbs. It is equipped with internal batteries and is capable of diving to depths up to 100 meters for as long as 22 hours on one charge. The vehicle is equipped with side scan sonar for its search function. Navigation is accomplished primarily by the vehicle ranging with in-stratum transponders. Secondary navigation is done by dead reckoning with an internal compass and Doppler velocity log. The navigation system is discussed in more detail in Chapter II. On-board computers store mission essential parameters and collect data for post-mission analysis. Data collected include salinity, temperature, depth, optical backscatter, side-scan sonar images and vehicle location information. The REMUS (in its transportation box) is pictured in Figure 3.

Figure 3. The NPS REMUS UUV in Transportation Container (8/03)

C. OBJECTIVES AND RESEARCH QUESTIONS

This research is concerned with the accuracy with which the vehicle estimates the location of an object. Greater uncertainty in the location results in more time for divers to clear an obstacle because of the increased search effort required. The amount of time a diver is allotted for the search effort is limited by several factors, including bottom time, the amount of air carried and diver fatigue. Decreasing the amount of time required to locate an object by shrinking the search area would save time, money and personnel.

Many components of the navigation system contribute to uncertainty about the vehicle's true location and thus the location of the contact. Deciding which components affect the error most severely and which might be influenced the most by human operators is the main area of interest for this thesis. Generating enough data from actual vehicle runs is prohibitive in terms of time. It would also be difficult to isolate individual components of the navigation system in an ocean environment. This thesis uses a Discrete Event Simulation (DES) model of the navigation that allows the analyst to explore how individual inputs contribute to position prediction error. This DES is implemented in SIMKIT and JAVA™. A more extensive description of DES and SIMKIT may be found in Chapter II of this thesis.

This thesis seeks to provide insight into three critical areas of vehicle operation. First, what operating conditions provide the best prediction of the probability of detection

of a mine like object? Second, given that detection has occurred, how can the predicted mean offset location error be minimized? And third, once a predicted mean error is determined how does the operator establish a probability that the mine is actually inside of a desired range. Insight into these three questions is provided by 1) building a model that predicts the probability of detection given certain input parameters, 2) building a separate model which predicts mean offset error given some input parameters and that detection has occurred, and 3) determining that the distribution of offset errors follows some known distribution that can be used to establish a probability that the mine is inside of the desired range.

Predicting the offset distance with accuracy is important because operators are concerned with the area that may have to be searched to clear the mine after the vehicle has found it. Many factors determine how long a team may have to search an area, such as bottom time for divers or mission requirements, so using search area affords the decision maker the ability to estimate time to complete the mission. The insight gained from analyzing the effect of combinations of inputs on output accuracy can help operators choose the best configuration for the vehicle prior to putting it into the water.

D. ORGANIZATION OF THESIS

Chapter II provides background on the various components of navigational accuracy in the vehicle and provides a primer in event graphs and DES. Chapter III explains the methodology for the simulation development. Chapter IV discusses the inputs to the simulation and analyzes the output of the various runs. Chapter V summarizes findings and explores follow on research questions.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. REMUS NAVIGATION DETAILS

The REMUS vehicle operates autonomously while performing its mission and thus navigation error is an important issue the operator must consider when analyzing the output from the mission. Understanding the vehicle's navigation methods and the potential sources of error provides insight into the output from mission playback and assists operators in interpreting collected data.

1. Methods of Navigation

REMUS uses three methods for navigation during a mission. The method can be pre-programmed by the user and is not required to remain constant throughout the mission. Mission design parameters may dictate that different methods should be used due to the vehicle's proximity to the transponders or by the ocean bottom composition, for example. REMUS may switch modes during the mission if it receives what are perceived to be bad fixes. The three methods, Dead Reckoning, Long Baseline, and Ultra-Short Baseline, are described below.

a. Dead Reckoning

If an acoustic fix is not available the vehicle navigates by Dead Reckoning (DR). DR position is computed using onboard speed and heading information. Speed information is calculated using propeller rpm input and Doppler acoustic signals. The Doppler acoustic signal may only be used if the vehicle is within 20 meters of the sea floor. Heading is computed using an onboard magnetic compass. DR position is computed by determining how far and in what direction the vehicle has traveled since the last update. This distance traveled is added to the last known position and the new position is updated in the computer. Precise DR navigation relies on accurate speed and heading information.

b. Long Baseline

With Long Baseline (LBL), the most common method of navigation, REMUS interrogates pre-positioned transponders and receives a return signal. The vehicle then triangulates its position based upon the length of time the signal takes to make the round trip. Many variables affect the speed of sound in water. REMUS collects

real time data on water temperature, salinity and depth to calculate the speed of sound and then calculates a distance based upon the time. Position is then fixed based on the triangulated range. LBL requires a minimum of two transponders. If a signal is not received from either one of the two transponders then the fix is assessed as "bad" and no position updates occur. The vehicle continues to navigate by Dead Reckoning between good. The vehicle may be set to interrogate at specified time intervals. The time interval affects the maximum range because a signal must depart and return prior to the next interrogation in order for the range to be accurate. Maximum range for the transponders is specified at 1500 meters, but ranges as long as 1700 meters have been observed to work in good water conditions. An example of the triangulation technique is illustrated in Figure 4.

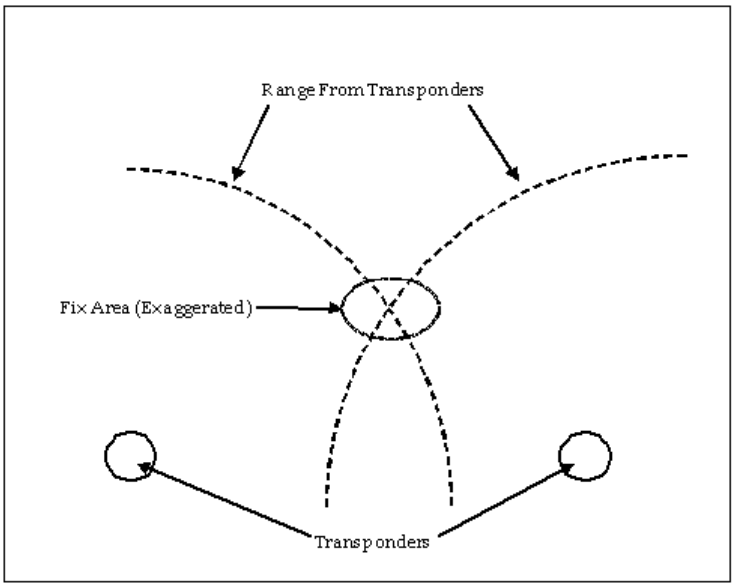


Figure 4. Triangulation Illustration With Two Transponders

Triangulation of position is achieved when the lengths of the three sides of a triangle are known. Figure 5 presents an application of the law of cosines to determine position [Stewart, 1999].

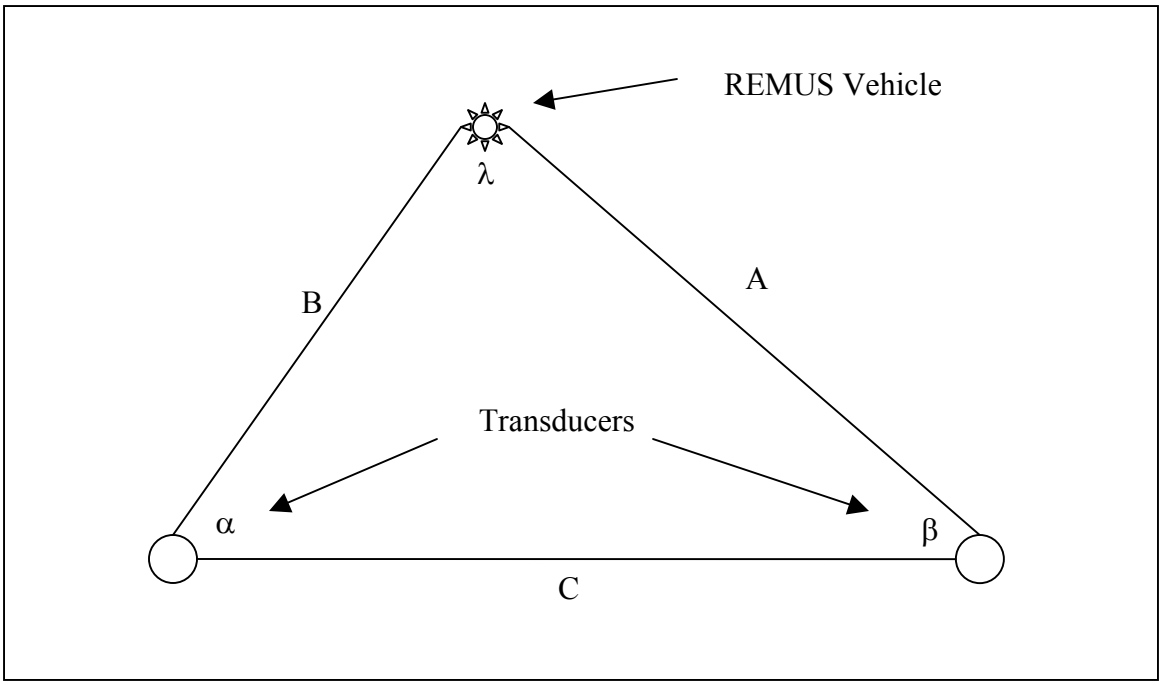


Figure 5. Law of Cosines Example

Distances A, B and C are known. The relationship between the sides and angle α is determined by Equation (1).

$$A^2 = C^2 + B^2 - 2CB\cos(\alpha) \tag{1}$$

Equation (1) is rearranged to solve for α as follows

$$\alpha = \cos^{-1}\left(\frac{A^2 - C^2 - B^2}{-2CB}\right) \tag{2}$$

REMUS's position relative to the transducer on the left is obtained by simple trigonometric identities as follows

$$\begin{aligned} x &= B\cos(\alpha) \\ y &= B\sin(\alpha) \end{aligned} \tag{3}$$

c. Ultra-Short Baseline

Ultra-Short Baseline (USBL) is used when the vehicle is finished with its primary mission and is preparing for recovery. The vehicle homes in on the signal from one of the transponders. This method is not used for navigation during field surveys and is not addressed further in this thesis.

2. Sources of Navigation Error

a. Compass Error

The compass onboard REMUS is the primary tool utilized during DR navigation. REMUS combines an ordinary magnetic compass with yaw-rate sensors to produce a fairly accurate heading input. The compass measures the absolute direction of the magnetic field to determine the vehicle's heading. In a compass only the horizontal component of the measurement gives the direction of north, thus the vehicle must be able to remove the vertical component caused by gravitational pull. Therein lies the main component of compass error because this vertical component is not constant. It is affected by pitch and roll changes in the vehicle and inaccurate measurement of the vertical component results in incorrect interpretation of the horizontal component, leading to heading inaccuracies. However, over the long term the accelerations tend to cancel one another out. For example, for every roll to port there is a corresponding (and effectively equal) roll to starboard. The net effect tends to be that in the long run the acceleration errors are effectively zero and the magnetic heading is correct. In the short term, however, there can be heading errors as high as 3%, or up to 10.8 degrees to either side.

The heading error is mitigated by use of a yaw-rate sensor. The yaw-rate sensor measures the rate at which the vehicle is turning. The rate is multiplied by the amount of time since the last measurement and the product is added to the last heading. The result is a fairly accurate heading. The sensor must have good heading information at the outset for this method to be successful. Additionally, small errors in measurement accumulate over time, thus the method is more accurate in the short term than the long term.

Combining a magnetic compass that is fairly accurate over long distances with the yaw-rate sensor that is more accurate over short distances produces a heading input with more accuracy than either would have produced individually. The manufacturer claims that heading errors are reduced to ± 0.1%, or about 0.36 degrees. This claim is difficult to verify and is not consistent with data collected in support of this thesis. The vehicle can only be operated in ocean environments because of the configuration of the ballast. Measuring the heading error would require that the vehicle be in an environment free from other factors influencing heading, such as current.

b. Effect of Current

The vehicle operates primarily in ocean environments where the effect of current can push REMUS off track or cause it to inaccurately assess its position on the planned track (Figure 6). The overall effect is to introduce error in the prediction of an object's position. When the vehicle approaches from the same orientation on consecutive runs over a short period of time the error is predictable because the current does not change direction or strength appreciably over short time spans or short distances. Current is not constant, however, nor is it easily determined at a specific location for use by the vehicle. Currents are often reported in a location in the vicinity of some important navigation aid but generally not in a specific location as might be necessary in order to apply correction to the vehicle ahead of time, thus the vehicle's position is subject to uncertainty due to the effects of current.

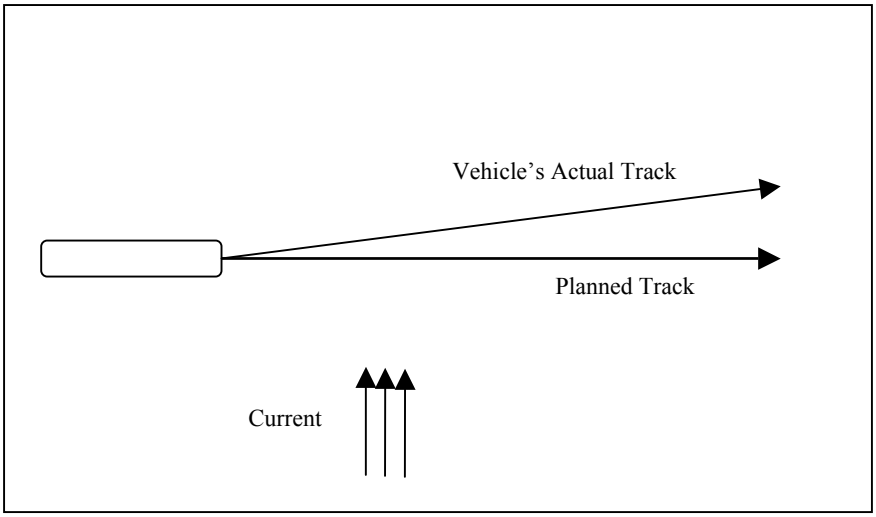


Figure 6. Effect of Current on Planned Navigation Track

c. Transducer Placement

Of all the variables, the operator has the most control over transducer placement. The transducer is transported to the drop area and positioned at the surface using GPS. When the operator is satisfied that the location is correct the transducer is dropped and allowed to sink to the bottom. It is held in place by an anchor attached to a neutrally buoyant line. The transducer is allowed to float in the same stratum as the vehicle. See Figure 7 for an illustration of transducer and anchor setup. The position of the transducer is programmed into the onboard computer of the vehicle and the computer

uses the location to triangulate its position. Generally the position is programmed into the vehicle prior to making the drop, but there is no restriction that would prohibit inputting the position after the drop occurs.

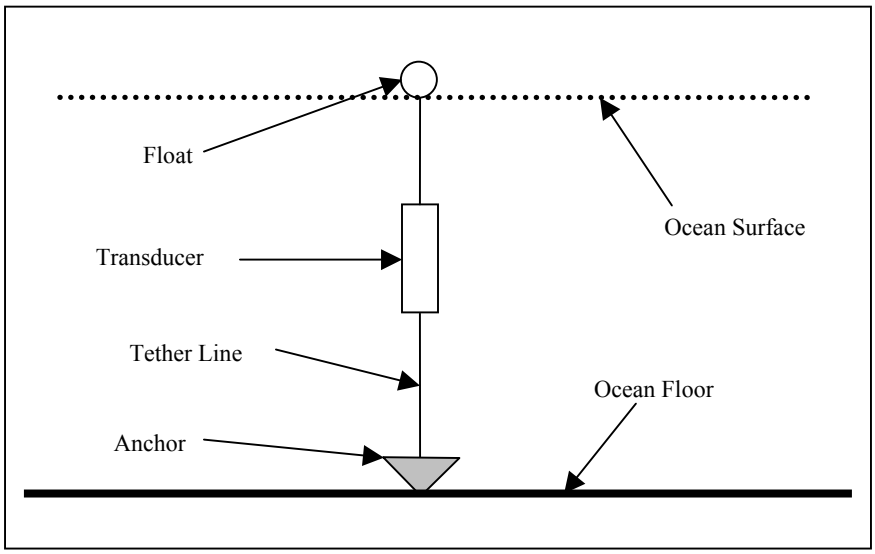


Figure 7. Typical Transponder Setup

There are two sources of error in transducer placement. First, the operator may not be at the exact position for the drop as input into the vehicle's computer. The accuracy of a handheld GPS unit is quantifiable and contributes to the misplacement of the transducer. This source of inaccuracy occurs as a one-time error for each transducer because of the assumption that once the device is dropped overboard it neither moves nor drags its anchor on the bottom. Secondly, the anchor and transducer do not drop straight down to the bottom. Current may affect the drop and the hydrodynamic effects acting on the device are unpredictable as it descends.

d. Transducer Motion

Once the transducer is placed on the ocean floor, it is affected by current and wave action surges, resulting in the most significant contributor to navigation error introduced by the transducers. The vehicle relies on accurate knowledge of transducer placement to triangulate its position. Under best case conditions a small displacement of the transducer results in a small error in position fix. However, these errors can be substantially magnified when the vehicle operates with a small angle between the baseline (an imaginary line joining the two transducers) and the vehicle's track. This

phenomenon is the result of fairly straightforward geometry considerations. Consider Figure 8 and Figure 9 for a visual explanation of this with only one transponder out of position. The results when both transducers are out of position are similar but more exaggerated.

Current has the effect of displacing the transducer from a vertical position off to one side with a fairly constant magnitude. The displacement is caused by hydrodynamic drag. Wave action tends to displace the transducer from side to side in a repeating manner. The overall effect is that the transducer is rarely in the position that the vehicle is programmed to utilize in its calculations. In the long run the effect of waves is mitigated by its alternating behavior, but in the short-term there can be substantial error.

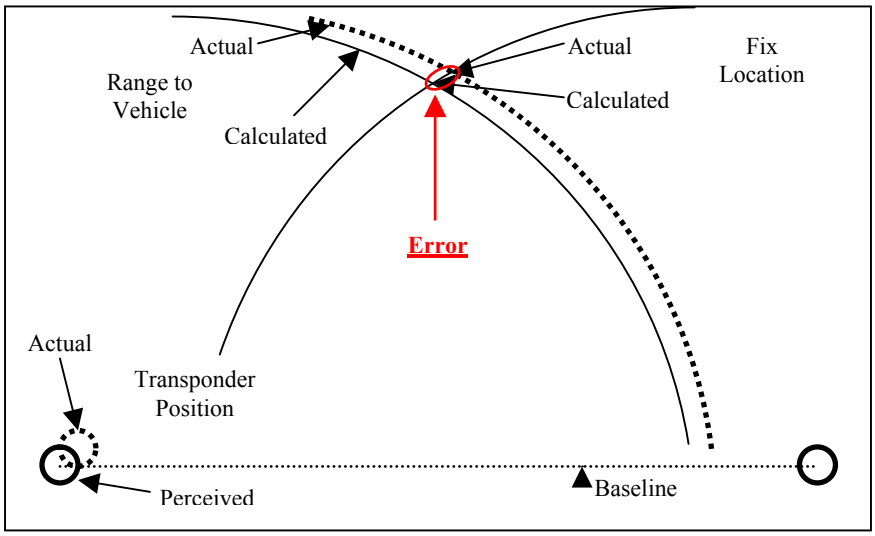


Figure 8. Possible Position Uncertainty for Large Angle

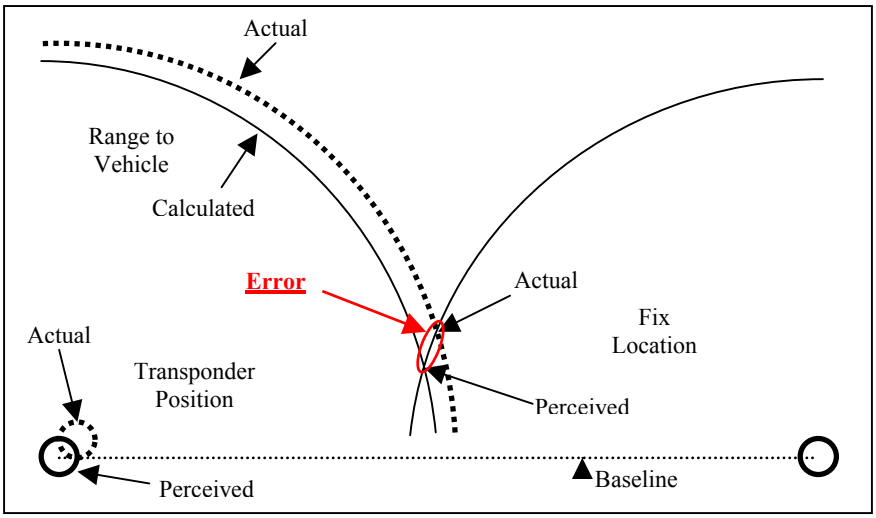


Figure 9. Possible Position Uncertainty for Shallow Angle

e. Vehicle Attitude

The final consideration for position uncertainty involves the vehicle attitude relative to the location of the transducers. Consider first the distance from the vehicle to the transducer. As the distance increases the likelihood that the transponder "hears" the interrogation of REMUS or that REMUS "hears" the return ping degrades. The physics of sound propagation in water are similar to those in air and the amount of energy that the vehicle and transponder put into the water do not change, thus the intensity of the sound decreases as range increases. The second consideration is the vehicle's relative aspect to the transponders. The vehicle is more likely to receive the transponders signal when it is pointing toward the transponder than when it is traveling away from the transponder due largely to the location and construction of the sensors in the vehicle's nose. Recall that a good fix is obtained, and thus an update calculated and stored in the on-board computer, only if a response is received from both transponders. The vehicle will navigate by Dead Reckoning between good fixes. The longer the vehicle goes without a good fix the higher the position uncertainty. This is explained and demonstrated more completely in Chapter IV of this thesis.

B. DISCRETE EVENT SIMULATION

The vehicle simulation described in Chapter III of this thesis utilizes Discrete Event Simulation (DES). Introductory remarks are made here to assist in grasping the fundamentals of the simulation. For a more detailed treatment of DES refer to any

introductory level text, such as Law and Kelton's *Simulation Modeling and Analysis*. The following general discussion is derived from Buss [2002] as well as Law and Kelton [2000].

1. State Variables and Event Lists

DES is based upon state variables and events serving as two fundamental components working together. State variables define the state of a system at any particular moment in time. More than one state variable may exist in the model (as is often the case) and the choice of how to define the state variables is critical in collecting meaningful data from the output of the simulation. The time-varying values of the state variables, referred to as state trajectories, are studied to draw conclusions about the performance of the system being modeled.

The state trajectories in a DES are piecewise constant; the value of a state variable does not change until some previously scheduled event occurs. Time does not progress during execution of an event, only during the period between execution of one event and another. Event execution is scheduled and processed by the event list. The event list stores future events and may be thought of as a "to-do" list for use by the program. An event notice is generated containing information about which event is scheduled to occur and at what time in the simulation it is scheduled to occur. This is perhaps the main distinction between discrete event modeling and time stepped modeling. In DES the time is advanced to the next scheduled event and that event is processed. In time stepped modeling the clock is advanced to the next time step (determined in advance) and if an event would have occurred during that time step it is then processed. It is not unlikely in time step simulation for the clock to "stop" several times before an event must be executed. In DES the clock only stops when an event is executed or the simulation stops.

2. Discrete Event Algorithm

It is helpful to view the process in the framework of the discrete event algorithm, borrowed from Law & Kelton [Law & Kelton, 2000]. In pseudo-code:

While (Event List not empty)

- Advance time to the next scheduled event
- Perform state transition for the event

- Remove the event from the list
- Schedule other event(s) (as required)

By convention, a predetermined order is employed when an event occurs. First, all state changes associated with the current event are performed. Second, any future events resulting from the current state changes are scheduled. Finally, the completed event is removed from the event list. The newly scheduled events are set by the current event and may be conditional on some other condition or state within the model. As Buss points out, it is possible to perform the above steps in an arbitrary fashion, but the result is at best confusion and at worst an erroneous model. Buss states, "There is considerable benefit from adapting a convention such as [the one described here]" [Buss, 2001].

3. Event Graphs

Event graphs provide a visual way to interpret the logic of the discrete event algorithm. The graphs have proven to be quite powerful in unraveling complicated systems and relationships within the systems. The basic idea behind event graphs is presented here. For more in-depth information the reader is encouraged to refer to Schruben's *Simulation Modeling with Event Graphs* [Schruben, 1983]. The description in the following section comes from Buss' paper. *Basic Event Graph Modeling* [Buss, 2001], which summarizes Schruben's work nicely.

Every event graph consists of at least two basic entities; nodes and edges. A node represents an event, or state transition, and an edge represents the scheduling of some future event. Figure 10 demonstrates a simple event graph for illustrative purposes.

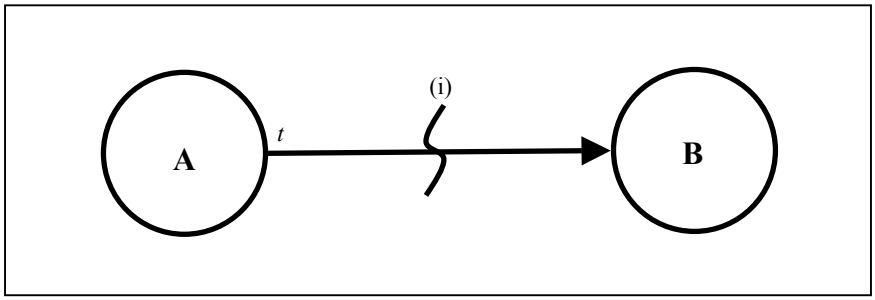


Figure 10. A Basic Event Graph (From Buss, 2001)

The interpretation of Figure 10 is as follows: The occurrence of event A causes event B to be scheduled following a delay of time t, providing that the boolean condition

(i) is met. Typically the time delay t will be some randomly generated number according to an underlying distribution with parameters set by the modeler. The Boolean condition (i) can be any prerequisite set by the modeler. For instance, consider the case of a single server queueing system. Event A may be the customer arrival event while event B could be the start service event for the customer. If the single server is busy then the customer must wait for service and event B will not be immediately scheduled.

C. SIMKIT

Simkit, developed by Buss at the Naval Postgraduate School (NPS), is designed with DES in mind. Simkit uses event graphs as its underlying methodology and is component based, meaning it adheres to the principle of object-oriented programming. What follows is a brief description of Simkit based on Buss' *Component Based Simulation Modeling with Simkit* (2002). Simkit is platform independent, written in the JAVA™ programming language, and is an open source package controlled under GNU public license. It is available for free download at http://diana.gl.navy.mil/Simkit/ at the time of this writing. Understanding of the remaining part of this section requires a working knowledge of the JAVA™ programming language; however, the remaining section may be skipped without loss of understanding later in this thesis.

1. The SimEntityBase

The *SimEntityBase* class is, as the name implies, the base class for almost all objects used in a discrete event simulation utilizing Simkit. Virtually all the functionality required for basic discrete event simulation is incorporated into the *SimEntityBase* class. The modeler implements model specific behaviors into sub-classes where parameters, such as random number generator parameters or transponder location information, are mapped into a read-only property. State variables, such as vehicle location, are also mapped into a read/write property in the sub-classes. State variables must be available to be changed by the program for update purposes.

Every event corresponds to a method with the same name except that a prefix 'do' is added (i.e. the event FixPosition becomes doFixPosition in the code). State transitions occur as changes to the corresponding instance variable in the code. In Simkit, a PropertyChangeEvent accompanies state transitions. This affords the analyst the opportunity to collect data from the simulation that might otherwise be lost.

2. Listener Patterns

The listener pattern allows designers to connect components together to create larger models of potentially great complexity. Simkit is designed to enable these connections without embedding the new components in already written and proven code, a very powerful concept. Two types of listeners are utilized in Simkit, the SimEventListener and the PropertyChangeListener.

a. SimEventListener

The SimEventListener pattern is an extremely useful tool that allows the modeler to add or change features of the model without making significant changes to the base set of code. It is generally used when an event in one component of the model is used to spur a corresponding event in another component. The listening component must be registered as a listener with the originating component and the signature of the event in question must be identical. The design of Simkit allows a component to be a source and/or a listener. Figure 11 demonstrates how the SimEventListener can be utilized.

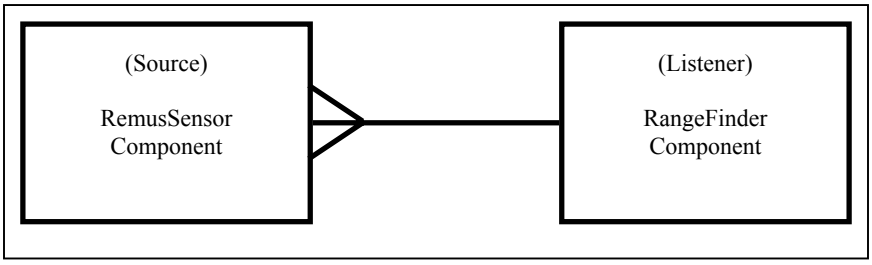


Figure 11. SimEventListener Event Graph Configuration

In Figure 11 the *RemusSensor* (discussed in Chapter III) is a source to the *RangeFinder* component. The lines connecting the two boxes represent the listening arrangement between the two components. Viewing the arrangement of lines as a notional stethoscope may help in understanding which component is listening to which.

b. PropertyChangeListener

Closely tied to the DES concepts of state and state transition, PropertyChangeListeners are key to efficiently collecting data from the model. Each PropertyChangeSource maintains a sort of list of all listeners that are interested in "hearing" when a state variable changes state. Again, Simkit was designed so that every *SimEntity* component may be either a source or listener for PropertyChanges. Notification

occurs when the component "fires" a PropertyChangeEvent. The source does not need to know what the listener intends to do with the information. In fact, it is not necessary for the listener to do anything with the information. The designer builds the components with a PropertyChange fired for every change in state with no thought needed for how the listening component may use that information. This allows subsequent users the flexibility to use existing code with no invasive modifications.

3. RandomVariate Generation

Generation of a reliable stream of random numbers is of interest to any modeler. Simkit provides a very robust set of *RandomVariate* objects. One problem facing any software designer is the impossibility of including every possible alternative to the end user. Simkit is designed to allow the user to generate new distributions as the need arises. An example of this occurred during the design of the first version of the REMUS model. A uniform circular distribution was desired for use in randomizing the effect of current on the vehicle. Simkit did not have a uniform circular distribution, but, with minimal effort on the author's part, a new class was written to implement this distribution. Random number generation is beyond the scope of this work, and detailed information can be found in Law & Kelton [2000].

4. Conclusion

Simkit enables the modeler to create flexible and reusable components without the need for invasive code writing. Once the underlying methodology is understood the process of building components is fairly simple and the modeler can quickly assemble complicated models. Simkit is ideally suited for the REMUS model precisely because of its flexibility and extensibility. More detailed descriptions of how different behaviors and processes of the REMUS vehicle were modeled are included in Chapter III.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SIMULATION DEVELOPMENT

A. BACKGROUND

The components making up the REMUS model represent the major pieces of the navigation system plus a few objects that "listen" for prompts so that they may collect data and otherwise conduct housekeeping functions. The full functionality of the model was not packaged into one large class, but was instead separated into classes based upon their use by the vehicle.

The components will be introduced in the order they are used. The execution of the simulation is broken down into steps. Initially, a simple model is built which incorporates only factors affecting Dead Reckoning, such as compass error and the effect of current. A more complex model is introduced which incorporates factors related to the LBL navigation method. Finally, the complete model is exercised in a typical mission involving sweeping a set area for a randomly placed mine. Later models incorporate the initial components and add functionality. Names of components are used to reference the name of the JAVA™ class in the model (e.g. *RemusMover* is a class name in JAVA™).

B. DEAD RECKONING MODEL

Dead reckoning is the default navigation method for the vehicle when it is not receiving good fixes and thus plays a part in every version of the model. The event graph displayed in Figure 12 is a representation of the model used to analyze the vehicle's navigation system while dead reckoning. The Dead Reckoning version models vehicle movement as uniform linear motion using the *RemusMover* class. A *CookieCutterSensor* models the vehicle's side scan sonar capability and data are generated using the *RangeFinder* component.

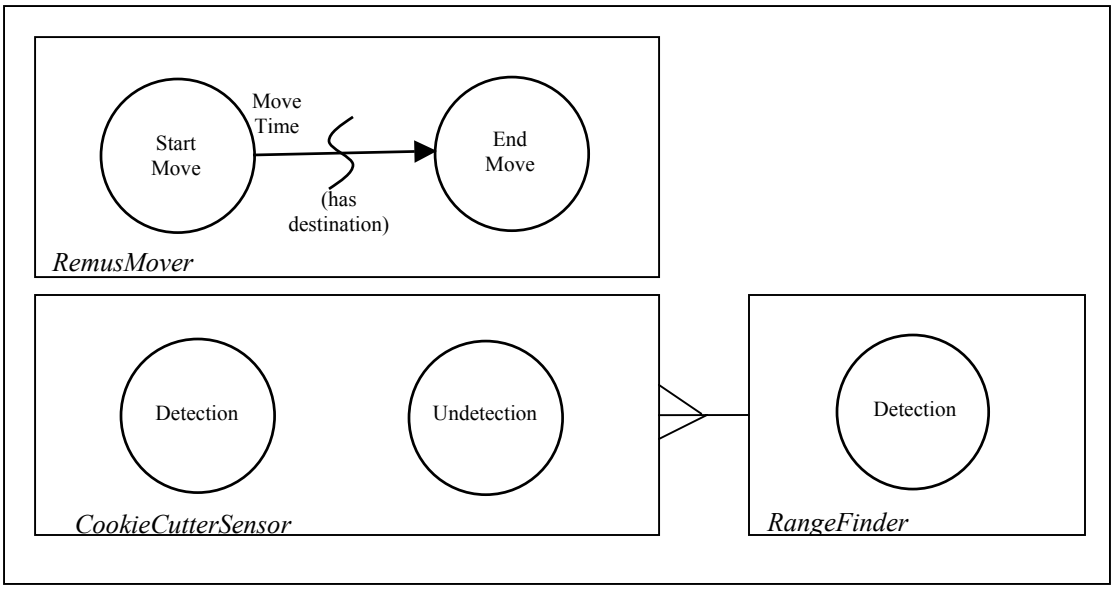


Figure 12. Event Graph for Dead Reckoning Model

1. RemusMover

The *RemusMover* component deals with how to monitor the vehicle's perceived position vs. ground truth while detecting obstacles. As the vehicle transits it keeps track of its position (consider it an imagined position) in the operating space. During mission playback the position information is accessible to the user. The position is not the vehicle's actual location, however. The effect of compass error and current act to push the vehicle off track and error is introduced. While collecting data from the model, it is useful to be able to ascertain the magnitude of the error. This is accomplished by keeping track of the vehicle's actual position and its imagined location, or where it thinks it is.

Movement is initiated by the StartMove event in the *RemusMover* class. All movement in this model is uniform linear motion, meaning that the vehicle is assumed to always travel from point to point in a straight line. This assumption is made in the interest of simplifying model implementation. Recall that in DES, time does not advance in regular intervals. Rather, time advances to the time of the next scheduled event on the event list. Changes in state (e.g. position, etc) are performed when the time reaches the next scheduled event. This presents a modest obstacle in the DES framework in that the position of the vehicle is constantly changing as it travels from point to point.

The obstacle is overcome by maintaining enough information about the vehicle so the position is not required to be kept explicitly, but rather can be maintained implicitly.

The *RemusMover* maintains three explicit state variables, position when movement began, time movement started and velocity, which together can be used to calculate the vehicle's position at any time between events. These variables are maintained for both the vehicle's "real" position and its "imagined" position (e.g., where its onboard computer thinks it is).

a. Uniform Linear Motion

Uniform linear motion is fairly straightforward to implement. Assume the moving entity begins its move from some initial position x_o at time t_o with velocity v. Note that these are the three explicitly maintained state variables mentioned earlier. The equation of the position of the entity at time t is given by

$$x(t) = x_0 + (t - t_0)v \tag{4}$$

The quantity given by Equation 4 is not computed continuously or even at regular time intervals as in time stepped simulation. It is instead only computed as needed by the simulation.

b. Mover Managers

The *RemusMover* component only executes orders to move from one point to another. It does not store or manage any information about the points it may be expected to move to in the future. Only the most recently departed point and the point it is moving to are maintained. Functionality is added by having a component that will manage the various points and coordinate the vehicles path. The *PathMoverManager* class accomplishes this task. Points are passed to the manager in the form of waypoints in the order in which the vehicle is expected to transit. In this way more complicated paths can be programmed that more accurately reflect desired behaviors for REMUS. The *PathMoverManager* class is a part of Simkit and is not modified for this model. The *PathMoverManager* listens for an EndMove event from the *RemusMover*. The listener pattern is established in the constructor for the mover manager. When the EndMove event is heard the manager checks for another waypoint and, if one exists, issues orders to the *RemusMover* to move to the next waypoint.

c. The Movement Order

When the manager directs the *RemusMover* to execute a move, several calculations occur that are critical to maintaining position information. The vehicle is ordered to move if the movement order is accompanied by an actual destination. This is represented by the conditional "has destination" on the scheduling edge between the StartMove and EndMove events of the *RemusMover* component of Figure 12. Time t to travel to destination y from initial location x with speed s is computed for the ideal condition where the vehicle knows its position with certainty as follows:

$$t = \frac{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}{s} \tag{5}$$

The time from Equation 5 is used to schedule the EndMove event for the *RemusMover* and is represented in Figure 12 by the moveTime parameter on the scheduling edge between the StartMove and EndMove events of the *RemusMover* component. The state variable for movement start time is set using the current simulation time. The amount of time required to move from initial to final location is used to calculate the movement for the "real" destination, as well. The main difference in the calculations lies in how the velocity is used. In the case of imaginary movement, velocity is used as supplied by the manager, but in the case of real movement, velocity is adjusted for compass error and the effect of current.

d. Compass Error Implementation

Compass error is passed to the *RemusMover* via the test program as a maximum percent error of 360^{o}. Each time the mover is ordered to move by the manager the compass error is calculated and the real velocity of the vehicle is adjusted. Ideal velocity remains unchanged. The random error required by the model is produced in the test class so that the distribution parameters may be controlled more easily without access to the code for the *RemusMover*. Distribution parameters are discussed in more detail in Chapter IV.

e. Current Implementation

The effect of current is also implemented through an adjustment to the real velocity while leaving ideal velocity unchanged. The current adjustment is passed to the *RemusMover* class via the test class in the form of a vector consisting of x and y

components that are altered by adding some randomly generated noise. The adjustment is made at the time a movement order is passed to the *RemusMover*. The vector is produced in the test class, again allowing distribution parameters to be controlled more easily. Distribution parameters are discussed in more detail in Chapter IV.

2. CookieCutterSensor

The sensor for the model is assumed to be a simple cookie cutter sensor, meaning that as soon as the obstacle is inside the range of the sensor detection occurs with probability 1. The detection range is adjustable. The sensor is attached to the *RemusMover* and thus has the same movement characteristics as the *RemusMover* component. The problem then becomes how to calculate the time that the obstacle enters the range of the sensor, t_d. Because the sensor is moving in a uniform linear fashion, the equation of the sensor's position also is given by Equation 4. Since detection occurs when the distance between sensor and target is exactly the sensor detection range R, the time of detection is the solution of

$$\|x_0 + t_d v\| = R \tag{6}$$

Expanding Equation 6, t_d is given by the solution to

$$\|x\|^2 + 2t_d x \circ v + t_d^2 \|v\|^2 = R^2 \tag{7}$$

Since t_d is now expressed in Equation 7 as a quadratic the solution(s) are given by

$$t_d = \frac{x \circ v}{\|v\|^2} \pm \frac{\sqrt{\|v\|^2 (R^2 - \|x\|^2) + (x \circ v)^2}}{\|v\|^2} \tag{8}$$

Five possibilities for the value(s) of t_d exist [Buss, 2003]:

- Both roots positive. The target will enter and exit the range. See case 1 in Figure 13 for an example. Point A represents the time of calculation. Point B is the time the target enters sensor range (smaller value) and point C is the time the target exits the sensor's range (larger value).
- One positive and one negative root. The target is already within the sensor's range and the target will exit the range. See case 2 in Figure 13

for an example. Point D represents the time of calculation. Point E is the time the target exits the sensor's range (the positive root).

- Both roots negative. The target is outside the range of the sensor and the sensor is moving away from the target. See case 3 in Figure 13 for an example. Point F is the time of calculation.
- No real roots. The target will never be inside the range of the sensor. See case 4 in Figure 13 for an example. The point G represents the time when the calculation occurs.
- One positive root. The sensor will pass the target at exactly the range of the sensor (a tangent).

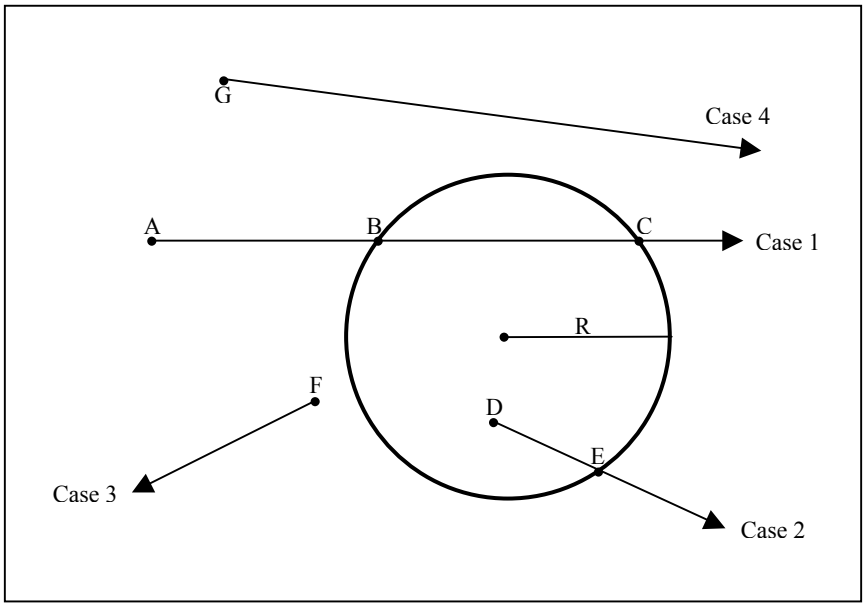


Figure 13. Possibilities for Cookie-Cutter Detection (After Buss, 2003)

The detection and undetection events from *CookieCutterSensor* in Figure 12 are scheduled by another class called *RemusSensorMediator*. Detection events are scheduled when the mover reaches the Closest Point of Approach (CPA) with the target, and undetection events are scheduled to occur when the mover exits the sensor's detection range. For clarification, detection is only scheduled if the target enters the sensor's range of detection, and then detection is scheduled to occur when the target and mover are at CPA. This behavior is necessary to model the way an operator locates obstacles in the

side-scan images during post-mission analysis. A property change is fired when a detection or undetection event is processed on the event list. These property changes can be used to gather statistics on the number of detections or times of detection, for example.

3. RangeFinder

The *RangeFinder* class gathers data on the error in target location by listening for a detection event in the *CookieCutterSensor* component. This is represented by the lines connecting the *RangeFinder* and *CookieCutterSensor* components in the event graph of Figure 12. The *RangeFinder* component has a detection event that is scheduled on the event list whenever the detection event of the *CookieCutterSensor* is scheduled. The detection event calculates the offset in predicted vs. actual target location and outputs the data to a text file designated by the user at run time.

4. RemusRunDR

RemusRunDR is the executable class that contains all the parameters and instructions for the dead reckoning data collection runs. In short, the components are instantiated, listening patterns are established, random variate parameters are set, experiment levels are established and replication loops are set up. More detail on the model inputs can be found in Chapter IV.

C. TRANSDUCER MODEL

The preferred method for navigation utilizes the transponders for triangulation of position. The vehicle still uses Dead Reckoning as its navigation method between fixes. Ideally the error in predicting obstacle location should be minimal using this improved navigation technique, and the magnitude of the error is indeed improved, but substantial errors still exist due to Transducer placement errors and movement of transducers after initial placement. The event graph for the model incorporating transducers is provided in Figure 14. Previously discussed conventions apply to this event graph. A new edge is introduced in the *RemusMover* component. The dashed line joining several events, such as the Fix and Intercept events, represents an interrupt action. The interrupt action removes the event on the head of the edge from the event list. More detail will be provided in the specific discussion of each event.

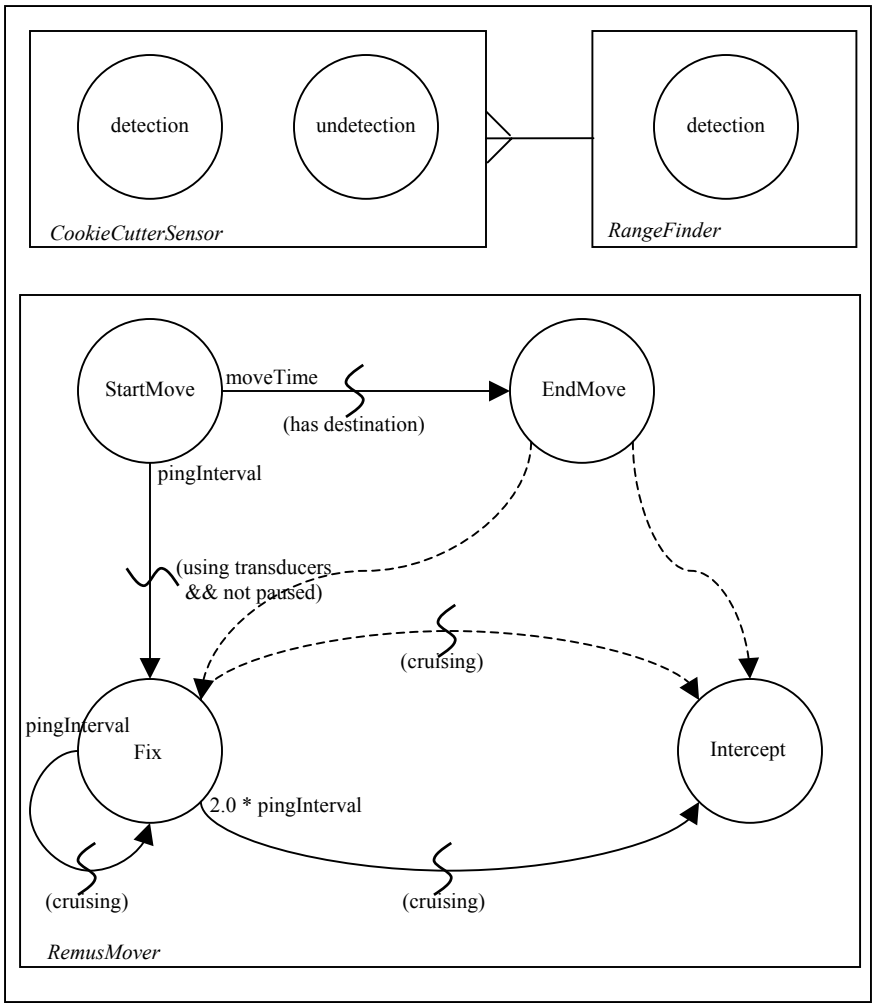


Figure 14. Event Graph for Transducer Model

1. Improved RemusMover

Many details of the *RemusMover* component remain unchanged. The same location information and movement procedures are followed in the transducer model as were used in the Dead Reckoning model. Compass error and current effects are implemented in the same manner. The major alteration to the *RemusMover* component involves the implementation of position fixes using the transducers. This capability was not present in the first version of the *RemusMover* because it is not necessary to have transducers for Dead Reckoning navigation.

a. The StartMove Event

The StartMove event must schedule the first Fix event if the vehicle isn't paused and the simulation is using transducers for navigation. The Fix event is scheduled after a delay determined by the ping interval variable. A longer ping interval means more

time will elapse between fixes and more error will accumulate between fixes. A shorter ping interval implies less error will accumulate, but that reduction in error comes at the expense of more computing time. Additionally, a shorter ping interval means that the vehicle cannot be operated as far away from the transducers because the vehicle would have to wait for too long for the return signal. The StartMove event also schedules the EndMove event providing that the vehicle actually has a destination assigned to it.

b. The EndMove Event

The EndMove event represents the end of the vehicle's current leg. The vehicle will no longer be moving, or will be assigned a new destination, and thus no longer requires a Fix event to be scheduled. The execution of the EndMove event interrupts any scheduled Fix event. This behavior is a simplification of the vehicle's actual method of obtaining fixes. In actuality the vehicle never stops attempting to fix its position, even while turning or maneuvering to the next waypoint. However, this simplification does not lead to a loss of generality. The time to execute a turn in the simulation is essentially zero and so the vehicle is fixing its position again immediately after the turn. The Intercept event is also interrupted by the execution of the EndMove event.

c. The Fix Event

Position fixes are developed and implemented in the Fix event. The Fix event is initially scheduled by the StartMove event, and is rescheduled when the Fix event is executed and certain conditions are met. The first logic gate determines whether a good signal was received from each of two transducers. As the distance between the vehicle and a transducer increases, the probability that a good signal is returned to the vehicle decreases. Appendix A contains the analysis conducted to determine the distribution of ranges. The Fix event determines if the range between the vehicle and each transducer is less than a number drawn from the exponential distribution. Both transducers must return good signals for the triangulation technique to work. The vehicle's position is calculated and the imaginary position is updated if both signals are good. Note that while the imaginary position is updated it is not necessarily the same as the vehicle's real position due to errors associated with the transducers. No position

update is entered if either one of the signals is not good. In either case the next Fix event is scheduled with a delay equal to the ping interval.

The vehicle drives back toward the original track once a position fix is executed. The vehicle's position is projected onto the original track and the vehicle alters course to intercept the track at a point two fix intervals farther down the track. An Intercept event is scheduled by the Fix event execution. Any prior Intercept events are interrupted by the Fix event, as well. Eventually the vehicle will be in a position where the ultimate destination is closer than the point calculated for intercept. In this case the vehicle is given orders to move to the final destination. All of these tasks are executed only if the vehicle is actually moving (determined by whether the vehicle's movement state is "cruising").

d. The Intercept Event

The vehicle could travel for any amount of time after a good fix prior to executing another good fix. In the interim the vehicle uses dead reckoning to navigate. A course alteration is executed if the vehicle reaches the track prior to the next fix occurring and the vehicle is ordered to move to the final destination. This behavior is handled by the Intercept event.

2. Transducer

The vehicle uses transducers to triangulate its position. Transducer behavior is implemented in the model by use of the *Transducer* component. Like the *RemusMover* component, the *Transducer* maintains a real and an imaginary location. The imaginary location is the spot where the vehicle thinks the transducer is located based upon pre-programmed information. The real location reflects position uncertainty generated by drop error and the effect of current and wave action.

Drop error accounts for uncertainties in operator use of GPS to position each transducer at the beginning of a run. In reality the operator positions the transducer at a location using a hand-held GPS unit and then drops it into the water so the anchor can rest on the bottom. Drop error is implemented in the *Transducer* component by adjusting the imaginary location by some amount and storing the new location as the real location. Once set, the drop may be altered at the beginning of each run by using a setter method provided in the code.

The effect of current on the transducer causes it to be offset from its imaginary location proportional to the intensity of the current and in the same direction as the current direction. Hydrodynamic theory is complicated and the detail possible in modeling this phenomenon exceeds the intent of this thesis, thus some simplifications are made[1]. The errors for transducer location are simplified to a bivariate normal distribution. The bivariate normal distribution is chosen for its ability to provide coverage in two dimensions and for the simplicity in adjusting the variance of the data in only one direction at a time. Refer to Figure 15 during the following discussion.

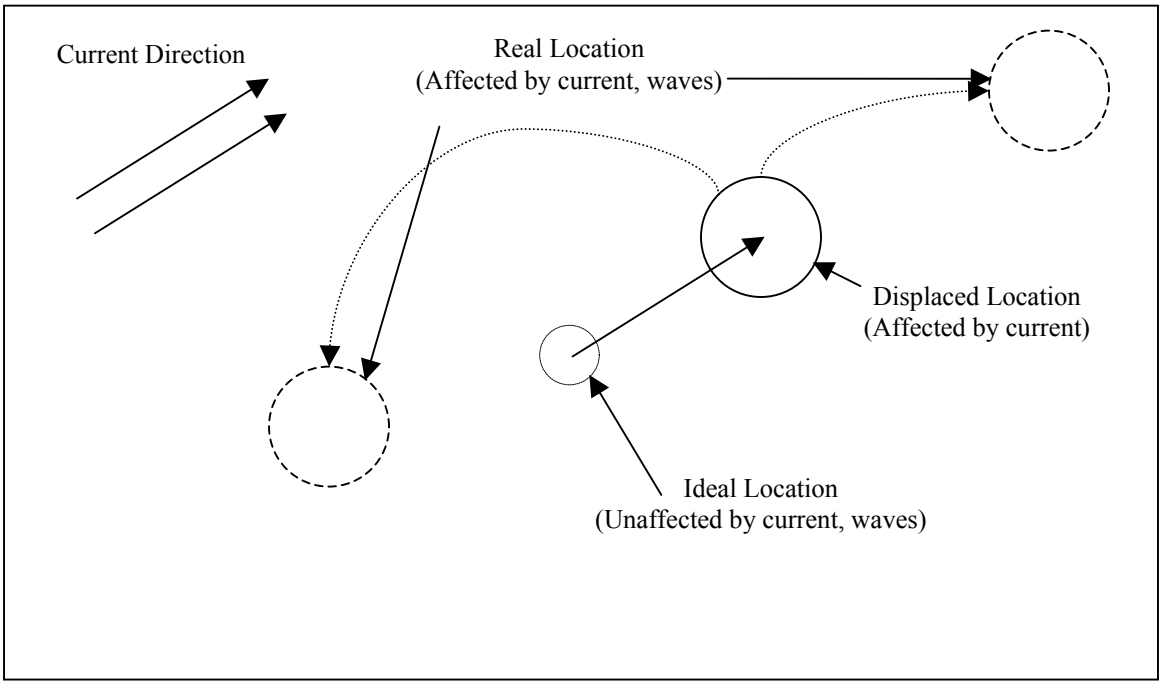


Figure 15. Transducer Current and Wave Action Implementation

Increasing current intensity tends to displace the transducer from its perfect vertical tending position in the same direction of current flow. The current direction is modeled by rotating the distribution so that the X axis is aligned with current direction. The current intensity is accounted for by shifting the mean of the X component of the bivariate normal more towards the positive side. Further increases in current intensity would shift the mean more, and reductions in current intensity serve to shift the mean less. Wave action is modeled by increasing the standard deviation of the X component of

[1] More information on the hydrodynamic effects of wave action on moored, submerged objects can be found in Verret's thesis, included in the list of references.

the bivariate normal. Higher wave height tends to cause more movement in the transducer and the standard deviation is larger. A calm sea with no current would produce a bivariate normal with both X and Y means of 0.0 feet and both X and Y standard deviations of 1.0 feet.

The bivariate normal random variate is passed to the *Transducer* as an argument and the position uncertainty is generated when the real location of the *Transducer* is queried. In this manner it would be possible to have current effects and wave effects change throughout a run, but that level of detail is beyond the scope of this thesis. The current speed and direction and wave height is set prior to each run and remains unchanged throughout the run.

3. CookieCutterSensor, RangeFinder

The *CookieCutterSensor* and *RangeFinder* components remain unchanged from the dead reckoning model.

4. RemusRunFinal

RemusRunFinal is the executable class that contains all the parameters and instructions for the transponder navigation data collection runs. More detail on model inputs is included in Chapter IV.

IV. MODEL INPUTS AND ANALYSIS OF OUTPUT

A. VERIFICATION AND VALIDATION INSIGHTS

Chapter III discussed how the model components are constructed and linked together, but the usefulness of the output of any model is tied to the reasonableness of the assumptions made regarding model inputs and the implementation of those assumptions in the logic of the code. Reasonable assumptions are generated by communicating with subject matter experts and collecting data on the system to be modeled in an attempt to establish patterns that may be of use. The REMUS navigation system model is largely an exploratory model in that little useable data exists to help formulate assumptions. Most assumptions were formulated based upon collaboration with the system operators.

1. Verification

Bratley et al. define verification as ". . . checking that the simulation program operates in the way that the model implementer thinks it does . . ." [Bratley et al., 1983]. Basically, verification involves checking that the code is bug free and accurately reflects the system being modeled. Simkit provides a verbose mode for this purpose. When enabled, the verbose mode outputs the current event as well as the future event list and the current status of each *Mover* component. The programmer is thus able to step through the simulation one event at a time and ensure all components are performing as planned.

The ability to check each event as it is executed is vital to ensuring that the more complex component interactions are being handled correctly by the DES algorithm. Each model version was checked in this fashion. Additionally, controlled scenarios were constructed and implemented to verify the logic. After each scenario was verified to execute properly, more complexity was added. Consider the following example of verification of the dead reckoning model.

The most elementary event is simply having the vehicle move from one point to another with no effect of current or compass error present. The vehicle had no sensor and no obstacle was present to detect. Additional points were added and the vehicle was verified to move in different directions correctly. A sensor was added to the vehicle and an obstacle was added to the scenario. The model was checked for proper EnterRange,

Detection, Undetection and ExitRange events. Compass error was introduced without randomness and the model was exercised through the scenario again. Finally the model incorporated the effect of current and was again cycled through the scenario. Only then was the model deemed ready for use. Similar techniques were used to verify the more complex versions of the simulation as those versions were developed.

2. Validation

Validation is not as easy. Bratley et al. define validation as ". . . checking that the simulation model, correctly implemented, is a sufficiently close approximation to reality for the intended application . . ." [Bratley et al., 1983]. This model is proposed as an exploratory model, meaning the output is intended to provide insights into possible uses and setups for the REMUS vehicle that cannot be explored by use of the actual vehicle due to time and cost constraints. Validation of this model would be difficult, at best, and in light of the intended purpose validation is not required, nor even desired. The model is useful as long as it operates in a logical fashion and the output provided "makes sense".

"Face validation" is a term used by Bratley, and others, to describe the process of validating assumptions using discussions with subject matter experts. This technique was utilized extensively while developing this vehicle simulation. For example, the effect of waves and current on transducer position would be very complicated to implement in a DES, and an argument can be made that the gain in realism is small and not worth the level of effort required in an exploratory model. The simplifications made in the implementation of transducer position error were discussed with experts in the Mechanical Engineering department with the conclusion that significant value is not lost. Similar discussions were conducted with vehicle operators at the Naval Postgraduate School to ensure that the simulation represented true vehicle behavior as closely as possible.

B. EXPERIMENTAL DESIGN

Understanding how the model inputs are related is important in discerning the relationships between model output data. Improperly designed experiments can lead analysts to draw erroneous conclusions. This model will not explore every possible combination of input factors exhaustively. Doing so would require prohibitively large amounts of computing time for the more advanced versions. However, limiting the inputs

to a couple of levels in each case might prevent complex interactions from being discovered. This experiment makes use of Latin Hypercube Sampling (LHS) to produce inputs that are not exhaustive but are detailed enough to uncover more complex behaviors [Box, 1978].

1. Terminology

The term *design* is used to denote a matrix whose columns correspond to *input factors*. An input factor is a specific model input parameter such as compass error or current direction, for example. The entries in the design matrix correspond to *levels* for each factor and each row of the design matrix represents a *design point* for the experiment.

2. Desirable Properties of the Design Matrix

In LHS each column of the design matrix is filled with different levels of a specific factor. The levels of the factor are evenly spaced and randomly sampled without replacement. Ideally the columns of the matrix would be orthogonal as evidenced by the pairwise correlation between columns being 0. In practice this is difficult, perhaps impossible, to achieve. Techniques exist to produce orthogonal design matrices but they tend to be restrictive in the sense that some levels may be excluded in order to achieve orthogonality. Unfortunately these conditions often cannot be known *a priori* [Kleijnen et al., 2004]. In light of this the objective becomes minimizing the pairwise correlation between columns.

Implementing LHS in the REMUS model is fairly straightforward. What follows is a brief description of the techniques used. Specific information for each version of the model is included in the appropriate section including samples of actual data. The first task is to produce a matrix that fulfills the requirement that each column be filled randomly and without replacement. The pairwise correlation between columns is calculated and compared to a threshold to screen out undesirable matrices. A threshold value for correlation of 0.25 was used to screen matrices, partly to reduce the number of candidates which must be produced. In a typical run approximately 60 matrices are produced from 10,000 candidates. The resulting matrices are then combined to form one large matrix used for the experiment. For example, if ten matrices are combined, each with ten design points, then the experiment will have 100 total design points. The effect

of combining the good matrices into one large design matrix is to lower pairwise correlations between columns. Typical values were no greater than 0.05 using this technique.

The remainder of this chapter is dedicated to developing three different simulation models. The models range in complexity from the relatively simple Dead Reckoning model to the most complex Area Sweep model. In each case the inputs to the model are discussed, the outputs from the model are described, and detailed analysis is conducted using output data.

C. DEAD RECKONING MODEL INPUT/OUTPUT ANALYSIS

Dead Reckoning is the most basic navigation method used by the REMUS vehicle. Data generated by the model are used to predict the probability of detection of an object given a combination of current speed and direction and to predict the mean location error given that detection has occurred. Simulation inputs and input assumptions are discussed first, followed by a description of output data format and content. A short introduction to the process of logit regression is provided to enhance understanding of the model parameters. Next, a logit model is built to predict the probability of detection given current direction and speed. Finally, a linear regression model is built to predict the offset error given that detection has occurred. Observations and conclusions on the Dead Reckoning model are presented to close this section out.

1. Inputs

Four factors are considered in this version of the model. They include maximum compass error, current direction, current magnitude and current noise. Of these four factors only compass error and current noise involve stochastic events. These are the factors that the operator has no control over when programming a mission. Current direction might be controlled by designing a mission to place the current in a desirable relationship with the vehicle, and similar results may be achieved with current speed by designing missions that take advantage of times when current intensity is small. Compass error and current noise cannot be realistically controlled by the operator.

a. Compass Error

Collecting data in an attempt to try to parameterize the actual compass error exhibited by an actual vehicle is cumbersome at best and nearly impossible at worst.

The main problem lies in the inability to separate the effect of compass error from that of current in the mission playback data. While historical current data are available the variance associated with observed currents typically is larger than the mean current value and thus is not useful in developing inputs for a simulation. Law and Kelton suggest that in the absence of data a heuristic approach can be employed to select an appropriate distribution [Law and Kelton, 1982]. The triangular distribution is proposed as an option. Intuition suggests that the error in compass heading is probably not equally likely to be 3% as it is to be, say, 0.5%. The triangular distribution produces numbers more often in the middle of the interval than at the edges and this behavior is consistent with the expected performance of the compass. A maximum error value is chosen to parameterize the random variate used to produce errors. The error is passed to the *RemusMover* component prior to the simulation starting and used to calculate how far off course the vehicle will drive. The error value is passed as a percentage of 360^{o}. For this model run the compass error factor is considered at eight different levels ranging from 0% to 3%.

b. Current Direction, Magnitude and Noise

Current direction is passed to the *RemusMover* with current magnitude and current noise in a package formed as a random vector consisting of x and y components. A new class named *UniformCircularVector* was written to implement this random vector component. Three arguments are passed to form this vector. The first two form the direction and magnitude of the desired current. The third establishes the maximum noise level. Refer to Figure 16 during the following explanation. The current direction and magnitude are passed to the *RemusMover* as an (x,y) vector as illustrated. The noise level is passed as a numerical value which represents the maximum noise level expected. The *UniformCircularVector* class produces a vector inside the circle centered at (x_o,y_o) with uniform density. This implementation provides for the ability to specify a prevailing current direction and magnitude and allows for some variation with a minimum number of parameters. Current direction, magnitude and noise are each considered at eight separate levels. Current direction varies at 45 degree intervals, current magnitude varied from 0 to 2 knots and current noise varies from 0 to 0.2 knots.

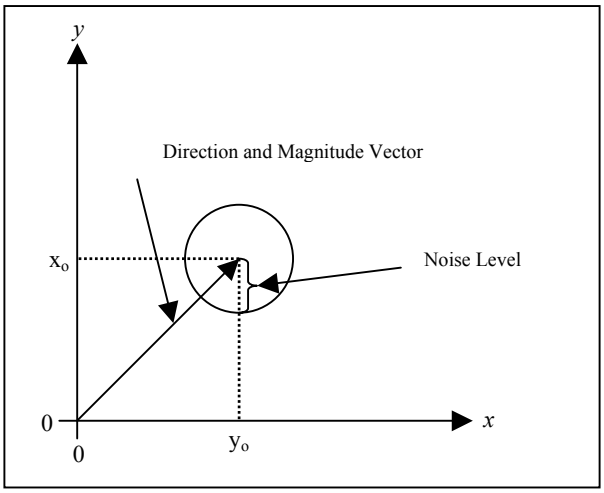


Figure 16. Current Input Illustration

c. Other Parameters

The sensor detection range is set to 35 meters, or approximately 115 feet. This reflects the actual limitation of the side-scan sonar onboard the vehicle. The vehicle will start from the origin, (x_o,y_o) = (0,0) with orders to move to (x_f,y_f) = (1000,0). All dimensions are in feet. The vehicle has a speed of 5 knots. An obstacle is placed at (x_m,y_m) = (500,0). These dimensions were picked in an attempt to mimic actual employment of the vehicle. No transducer field was utilized for this model.

d. Design Points

Table 1 provides the first eight design points for the DR run. 100 matrices were produced and combined for a total of 800 design points in the DR run. Using this many design points may produce some repetition but the benefit in terms of space filling is useful. In general, each factor is equally spaced between the lowest and highest permissible value. An exception is made in the case of current direction. Note that the case when direction equals 000^o is effectively equal to 360^o and so the upper level of current direction is fixed at 315^o to prevent inadvertent duplication. Pairwise correlations between the columns of the design matrix are given in Table 2. Combining the 100 matrices has reduced the maximum pairwise correlation between columns of the complete design matrix to less than 0.02 in all cases.

Current Direction	Current Speed	Current Noise	Max Compass Error
180	0.8571	0.1071	0.0214
135	0.0000	0.0714	0.0257
225	0.5714	0.2500	0.0000
270	0.2857	0.1429	0.0171
45	1.1429	0.0357	0.0043
315	1.7143	0.0000	0.0086
0	1.4286	0.2143	0.0129
90	2.0000	0.1786	0.0300

Table 1. First Eight Design Points for DR Run

	Current Direction	Current Speed	Current Noise	Max Compass Error
Current Direction	1.000	0.008	-0.005	0.005
Current Speed	0.008	1.000	0.019	0.010
Current Noise	-0.005	0.019	1.000	-0.015
Max Compass Error	0.005	0.010	-0.015	1.000

Table 2. Design Matrix Pairwise Correlation for DR Run Number One

2. Output Analysis

The output data file for each run contains the following information (data file header in parenthesis):

- Obstacle predicted *x* location (MineX)
- Obstacle predicted *y* location (MineY)
- Vehicle real final *x* location (VehX)
- Vehicle real final *y* location (VehY)

Also included for each repetition is the level used for the four factors (data file header in parenthesis):

- Prevailing current direction (CurrentDirection)
- Prevailing current speed (CurrentSpeed)
- Maximum current noise (CurrentNoise)
- Maximum compass error (maxCompassError)

Once the data are collected and imported into a data analysis program, the distance between predicted and actual obstacle locations is calculated (Offset) and a marker is assigned to reflect whether detection occurred on that run (denoted by a numerical value in the "MineX" position) or not (denoted by an "NA" in the "MineX" position). An example of the program raw output is provided in Table 3.

MineX	MineY	VehX	VehY	Current Direction	Current Speed	Current Noise	Max Compass Error
504.24	51.75	991.51	-103.75	270	0.8571	0.1429	0.0171
495.75	78.66	1008.79	-162.84	270	0.8571	0.1429	0.0171
NA	NA	1023.06	-254.69	270	0.8571	0.1429	0.0171
509.49	51.57	981.17	-102.3	270	0.8571	0.1429	0.0171
NA	NA	978.69	-251.24	270	0.8571	0.1429	0.0171
491.78	106.4	1017.57	-227.36	270	0.8571	0.1429	0.0171
NA	NA	1004.79	-244.69	270	0.8571	0.1429	0.0171
NA	NA	1004.04	-257.01	270	0.8571	0.1429	0.0171
498.41	62.76	1003.23	-127.98	270	0.8571	0.1429	0.0171
492.33	40.09	1015.67	-81.96	270	0.8571	0.1429	0.0171
503.71	81.51	992.42	-166.32	270	0.8571	0.1429	0.0171

Table 3. Subset of Output for DR Run One

Runs without detection are expected because the vehicle will, under the right circumstances, travel on a path which never intersects the detection range of the sensor.

a. Probability of Detection Given Current Speed and Direction

Of primary interest is whether or not the vehicle will detect an object given a set of operating conditions which the operator has control over, such as the current direction and speed. As discussed previously, the operator has control over current direction and speed to the extent that the vehicle can be operated to take advantage of existing conditions. A linear regression is not appropriate for modeling the probability of detection because predicted probabilities could be less than 0 or greater than 1. Logit regression provides a more realistic model for probabilities than linear regression. The following brief discussion of logit regression is derived from Hamilton [Hamilton, 1992].

The probability that a {0, 1} Y variable equals 1 is given by the expression $P(Y = 1)$. Given that Y is a dichotomous variable, the probability that Y equals 0 is given by $P(Y \neq 1)$, which is equivalent to $1 - P(Y = 1)$. The *odds* favoring $Y = 1$ are

$$O(Y=1) = \frac{P(Y=1)}{1-P(Y=1)} \tag{9}$$

Since probabilities lie in [0,1] it follows that the odds range from 0 to ∞. As an example, consider an event with a probability of 0.2. The odds that Y equals 1 are given by

$$O(Y=1) = \frac{P(Y=1)}{1-P(Y=1)} = \frac{0.2}{1-0.2} = 0.25 \tag{10}$$

These odds could be stated as 0.25 to 1 that Y equals 1, or similarly, 1 to 4 odds, which may be more familiar to most readers. The *logit* is obtained by taking the natural log of the odds as follows

$$L = \ln(O) = \ln\left(\frac{P}{1-P}\right) \tag{11}$$

Logits fall in the range (-∞,∞). *Logit regression* refers to a model with the logit as the dependent variable and the factors as the independent variables

$$L = \beta_0 + \beta_1 X_1 + \text{L} + \beta_{K-1} X_{K-1} \tag{12}$$

Since the logit L is a linear function of the X variables, probabilities follow a non-linear, s-shaped distribution as shown in Figure 17.

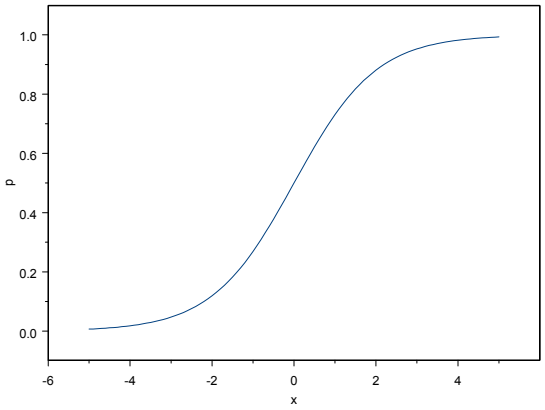


Figure 17. Logit Function

An expression for the probability of an event occurring with some logit value is given by[2]

$$\hat{P} = \frac{1}{1 + e^{\hat{L}}} \tag{13}$$

This expression is useful for graphing the probability of an event based upon some combination of the factors. As Hamilton points out, ". . . logit regression provides a more realistic model for probabilities than does linear regression" [Hamilton, 1982].

Skewed response variable distributions pose a problem for the logit regression. In the context of the DR model the occurrence of independent variables with a constant probability of detection regardless of the level of other factors can cause problems for the logit and add little predictive value to the model. If the points can be identified in advance then a claim can be made about the probability of detection without knowledge of any other factor level. Those points can then be discarded from the subset used to build the model. It is important to note that those data points are not discounted; they are simply not used to build the model. The information provided by those discarded points is valuable but no model is needed to predict the outcome at those points.

An investigation of Table 4 uncovers several such points. Observe that in all cases a current direction of 0 leads to detection regardless of current speed, current noise or compass error. Similarly, the case when current speed is 0 always results in detection, and while it is not the case that detection always results when current speed is 0.2857 the probability is very small that no detection will occur. The conclusion is that if current direction is 0 or current speed is less than 0.5714 knots detection will occur with probability 1. Eliminating these data from the model formulation in advance will enable a closer examination of the partial effects of the various remaining points.

[2] Hamilton actually gives this expression with a negative sign in front of the L in the denominator. The sign reversal in Equation (10) results from the way that JMP outputs the regression information, namely all coefficients are presented with a sign reversal.

		No Detect							
		Current Speed							
		0	0.2857	0.5714	0.8571	1.1429	1.4286	1.7143	2
Current Direction	0	0	0	0	0	0	0	0	0
	45	0	0	2	14	34	52	122	244
	90	0	1	13	38	420	681	1466	1799
	135	0	0	8	35	306	659	1269	1354
	180	0	0	0	0	1	4	18	19
	225	0	2	4	70	141	886	702	1565
	270	0	0	22	121	596	1124	1165	800
	315	0	0	1	0	37	76	168	443
		Detect							
		Current Speed							
		0	0.2857	0.5714	0.8571	1.1429	1.4286	1.7143	2
Current Direction	0	800	1400	1400	1700	1200	1700	1100	700
	45	1200	1700	998	986	1866	948	978	856
	90	1500	999	1087	1162	680	119	34	1
	135	1500	1000	1192	965	1094	441	131	46
	180	1100	1300	1000	1500	999	1096	1782	1181
	225	1800	1098	1096	1430	559	514	98	35
	270	1100	1400	1478	1279	804	76	35	0
	315	1000	1100	1699	700	1263	1624	932	957

Table 4. Current Direction, Current Speed vs. Detect for DR Model

The next important observation to make is that the event when current direction is 45 degrees is effectively the same as when current direction is 315 degrees in terms of the effect on the vehicle. In both cases the current is acting to push the vehicle off track either left or right and push the vehicle forward. Similarly, a current direction of 135 degrees is effectively identical to a current direction of 225 degrees and 90 degrees is effectively identical to 270 degrees. The result of this observation is that the output data can be aggregated so that fewer variables must be fit.

The logit model in its simplest form is obtained by regressing the four main effects on the logit. The summary for the DR model output is given in Table 5.[3]

[3] This output summary, and others of similar format, were obtained from the JMP Statistical Discovery Software, a product of the SAS Institute Inc.

Ordinal Logistic Fit for detection

Whole Model Test

Model	-LogLikelihood	DF	ChiSquare	Prob>ChiSq
Difference	20968.913	6	41937.83	0.0000
Full	11581.840			
Reduced	32550.753			

RSquare (U)	0.6442
Observations (or Sum Wgts)	52200

Converged by Gradient

Lack Of Fit

Source	DF	-LogLikelihood	ChiSquare
Lack Of Fit	444	1718.397	3436.794
Saturated	450	9863.443	Prob>ChiSq
Fitted	6	11581.840	0.0000

Parameter Estimates

Term	Estimate	Std Error	ChiSquare	Prob>ChiSq
Intercept[1]	-16.311772	0.2047565	6346.4	0.0000
currentSpeed	6.08461558	0.0640149	9034.5	0.0000
currentNoise	0.66080402	0.200378	10.88	0.0010
maxCompassError	27.6201506	1.7439096	250.84	<.0001
CurrentOff[45-0]	3.1104222	0.1600062	377.89	<.0001
CurrentOff[90-45]	5.52951396	0.0636285	7552.2	0.0000
CurrentOff[135-90]	-1.1208929	0.04099	747.78	<.0001

Table 5. DR Logit Model Summary Output

In all regression models investigated in this thesis, a p-value of < 0.05 is considered statistically significant. The R-squared(U) value from the logit model does not have the exact meaning that it does in the linear regression models that follow in later sections. However, the R-squared(U) value does measure the proportion of uncertainty that can be accounted for by the model and thus can be used to gain insight into improvements brought about by introducing more complex behaviors into the model. In general, a higher proportion of uncertainty explained by the model is good and so larger R-squared(U) values are desirable.

All of the coefficients are statistically significant in this first model, meaning that all four factors have some influence on the detection prediction. The coefficient magnitude can be misleading at first inspection until the range of the variable is taken into account. For instance, the coefficient for compass error is an order of

magnitude larger than the coefficients for some of the current offsets and the current noise coefficient. However, the range of data for compass error is only [-0.03, 0.03] thus the contribution by that variable is actually smaller than the other coefficients. The current offsets are modeled as categorical variables, meaning they either do or do not take on the indicated value. The base case, denoted by the intercept value in Table 5, occurs when current offset is 180 degrees, or pushing directly against the vehicle's direction of motion. The coefficients make sense. For example, the positive value of the current speed coefficient implies that as speed goes up the odds of a detection fall and thus the probability of detection falls, as well. The coefficient for current direction of 90 degrees has taken on a positive value implying that detection is less likely to occur when the current is pushing the vehicle from the side, and this makes intuitive sense.

Now consider the logit model which incorporates interactions between the main factors displayed in Table 6. This model results from performing a stepwise search with refinement and is the best fitting model for the Dead Reckoning data set.

Ordinal Logistic Fit for detection

Whole Model Test

Model	-LogLikelihood	DF	ChiSquare	Prob>ChiSq
Difference	22228.686	14	44457.37	0.0000
Full	10322.067			
Reduced	32550.753			

RSquare (U)	0.6829
Observations (or Sum Wgts)	52200

Converged by Objective

Lack Of Fit

Source	DF	-LogLikelihood	ChiSquare
Lack Of Fit	436	458.625	917.2492
Saturated	450	9863.443	Prob>ChiSq
Fitted	14	10322.067	<.0001

Parameter Estimates

Term	Estimate	Std Error	ChiSquare	Prob>ChiSq
Intercept[1]	-15.000043	0.238285	3962.7	0.0000
currentSpeed	7.55135437	0.1117908	4562.9	0.0000
currentNoise	1.30736682	0.371783	12.37	0.0004
maxCompassError	108.667807	3.227542	1133.6	<.0001
(currentSpeed-1.29885)*(maxCompassError-0.01483)	-264.1785	8.3571653	999.26	<.0001
(currentNoise-0.12267)*(maxCompassError-0.01483)	-43.251512	22.533727	3.68	0.0549
CurrentOff{0&45-90&135}	-3.7365348	0.0690452	2928.7	0.0000
CurrentOff{0-45}	-1.5218031	0.0808689	354.12	<.0001
CurrentOff{90-135}	0.68468091	0.0241039	806.87	<.0001
(currentSpeed-1.29885)*(CurrentOff{0&45-90&135}+0.1341)	-0.8379764	0.0737152	129.23	<.0001
(currentSpeed-1.29885)*(CurrentOff{90-135}-0.00766)	0.53787719	0.082903	42.09	<.0001
(currentNoise-0.12267)*(CurrentOff{0&45-90&135}+0.1341)	1.13035558	0.5399434	4.38	0.0363
(currentNoise-0.12267)*(CurrentOff{0-45}+0.14176)	-2.2562103	1.0505235	4.61	0.0317
(currentNoise-0.12267)*(CurrentOff{90-135}-0.00766)	1.21060137	0.2430166	24.82	<.0001
(maxCompassError-0.01483)*(CurrentOff{0&45-90&135}+0.1341)	127.426911	3.7728336	1140.7	<.0001

Table 6. DR Logit Model with Interactions Summary Output

Note that the R-squared(U) value has climbed to 0.6829 implying that more uncertainty is explained by this model. The trade-off is a sharp increase in the complexity of the model. There are many interactions to account for. Also note that even though the p-value for the noise:compassError term is above the threshold of 0.05 it is retained in the model. The interaction is retained because the model suffers without it and the p-value was very close to the threshold. Several other variations of the model were explored, including versions that took into account quadratic effects of speed and versions that employed data transformations such as the natural log of speed to try to fit

the data more completely but no attempt improved on the R-squared(U) obtained in this model. A contour plot of the predicted probabilities given current speed and current offset is presented in Figure 18.

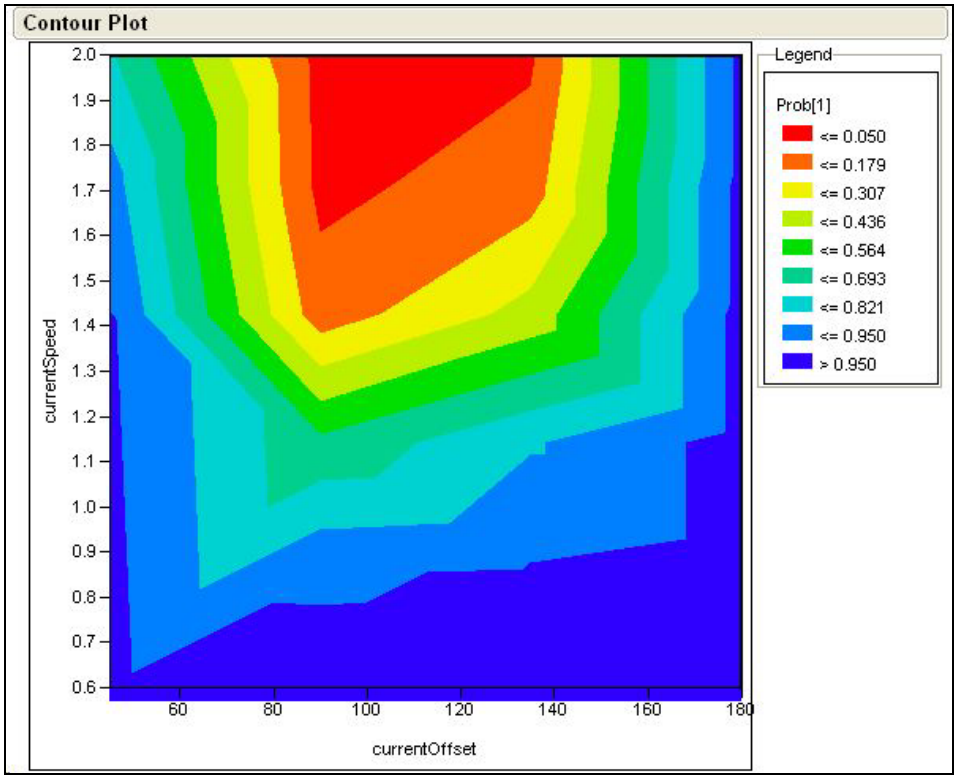


Figure 18. Predicted Probability of Detect Given Current Speed, Current Offset for DR Model[4]

As expected the probability of detection drops as current speed increases and probability drop-off occurs most rapidly when the current offset is approximately 90 degrees, or when current is perpendicular to the vehicle track and acting to push the vehicle off track. Predicted probabilities are acceptable with current speeds as high as one knot under most current directions. It is obvious that operating the vehicle in low intensity currents is desirable, but if current intensity cannot be minimized then the vehicle should be operated with the current acting to push the vehicle from behind or from ahead. Deciding between these two alternatives is best explored by considering a model of the magnitude of location error given that detection has occurred.

[4] This contour plot was derived using predictions from the applicable model.

b. Magnitude of Error Given Detection Occurs

The same original output data are utilized for the magnitude of error model with one modification. Since this model seeks to predict mine location offset given that mine detection has occurred, all data entries corresponding to non-detection have been eliminated. Thus the subset of data used only considers repetitions in which detection has occurred. This leads to some interesting patterns in the data that must be considered before attempting to build the model. Consider the plot of the predicted MineX and MineY locations in Figure 19.

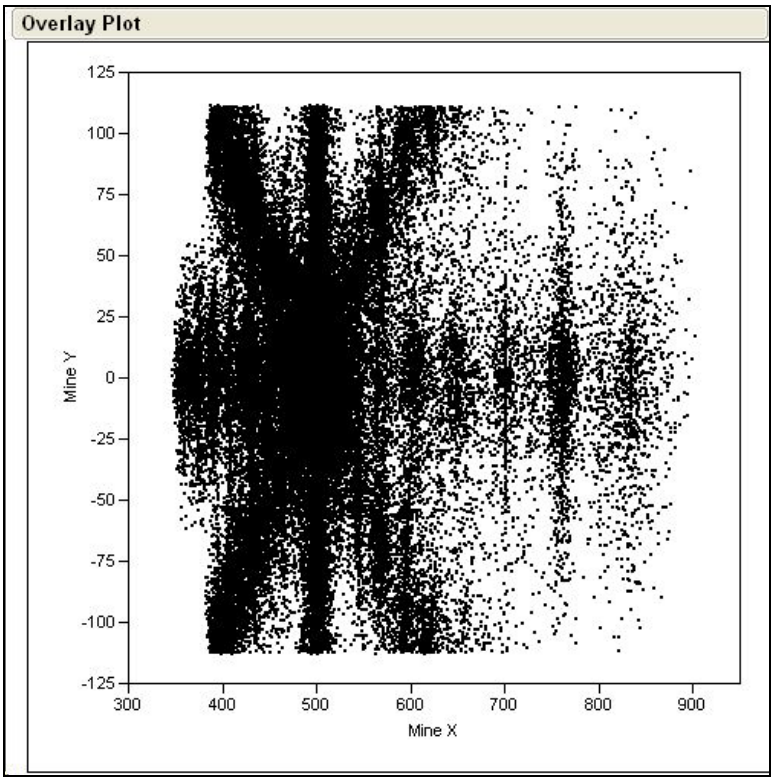


Figure 19. Plot of Predicted MineX and MineY for DR Run

First, note that the upper and lower limit for the MineY prediction is 115 feet. This corresponds to the maximum range of the cookie cutter sensor and represents a truncation point in data collection. Secondly, note that there are "stripes" in the plot. These stripes result from the way design points were constructed. The levels of design points were varied in discrete increments for simplification, but in reality the levels are continuous. Introducing more granularity in the model design points would result in more even space filling in the output data but would cost more in terms of computational

complexity. Finally, recall that the mine was placed at (X, Y) = (500, 0) for the DR model runs. The MineY distribution is effectively symmetric about the Y axis but the MineX distribution has a heavy right tail. See Table 7 for more detail.

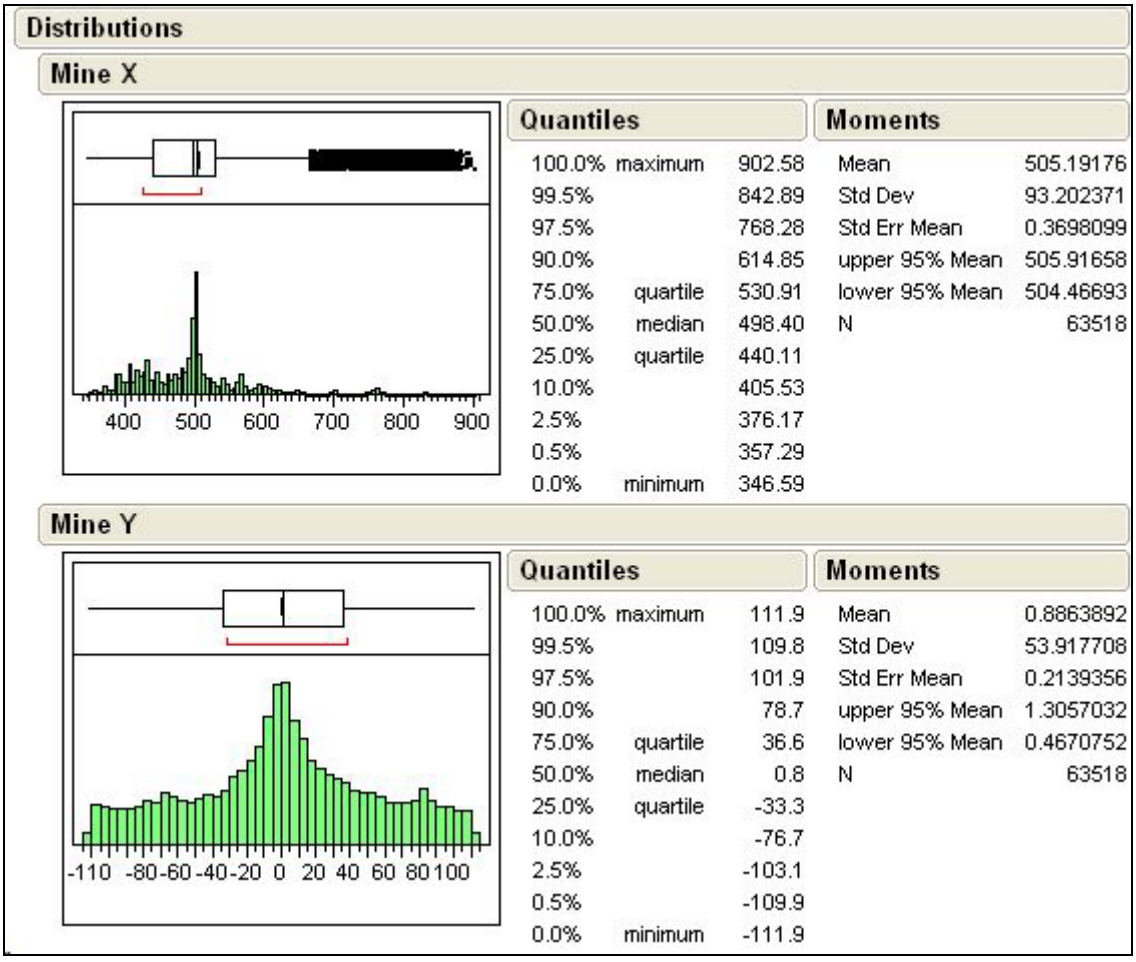


Table 7. Summary Statistics for DR Run Mine X and MineY

Predicted MineX locations to the right of the mine's actual location occur when the current acting on the vehicle has some component pushing against its direction of motion. More time elapses before the vehicle comes within range of the mine, thus more error develops and the predicted MineX value takes on larger values than is the case when current pushes the vehicle from behind.

Of interest to the operator is how to predict the position location error given information about the current speed and the direction of current flow relative to the vehicle's track. While the operator would nominally be interested in the compass error and current noise parameters it is unlikely that information would be available. The first

interesting model is thus the linear model implementing the two main effects, current offset and current speed. Table 8 summarizes this first model.

Response Mean

Weight: N

Whole Model

Summary of Fit

RSquare	0.872109
RSquare Adj	0.871211
Root Mean Square Error	214.427
Mean of Response	85.94858
Observations (or Sum Wgts)	63014

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	5	223238176	44647635	971.0451
Error	712	32737015	45978.953	Prob > F
C. Total	717	255975191		0.0000

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-5.832344	2.60607	-2.24	0.0255
CurrentOffset[45-0]	1.5777167	2.667637	0.59	0.5544
CurrentOffset[90-45]	10.460806	2.614448	4.00	<.0001
CurrentOffset[135-90]	12.517236	2.743341	4.56	<.0001
CurrentOffset[180-135]	43.119182	2.933238	14.70	<.0001
currentSpeed	89.692972	1.515661	59.18	<.0001

Table 8. DR Linear Model Output Predicting Mean Location Offset w/ Main Effects

Current offset is a categorical variable in this model. The base case, represented by the intercept condition, is when current direction is zero degrees relative to the vehicle, or pushing directly from behind. The current offset of 45 degrees was the only variable that was not statistically significant. All of the coefficients make sense. For example, the coefficient for current speed implies that as speed increases the predicted offset will also increase, and this make intuitive sense. Additionally, when current offset is at 180 degrees, or pushing directly from ahead, the predicted offset increases. This relationship is supported by the plots of MineX position from Table 7 which indicated that errors increased when current pushed from ahead. The adjusted R-squared value of 0.87 indicates a large amount of the error involved in prediction is accounted for by the model. Overall this is not a bad first attempt, but a closer look at the plot of actual vs. predicted mean offset presented in Figure 20 uncovers a weakness in this model. Observe that the model tends to underestimate at low and high offsets and overestimate at mid-range offsets. The shape of the plot suggests that some non-linear effect may be involved.

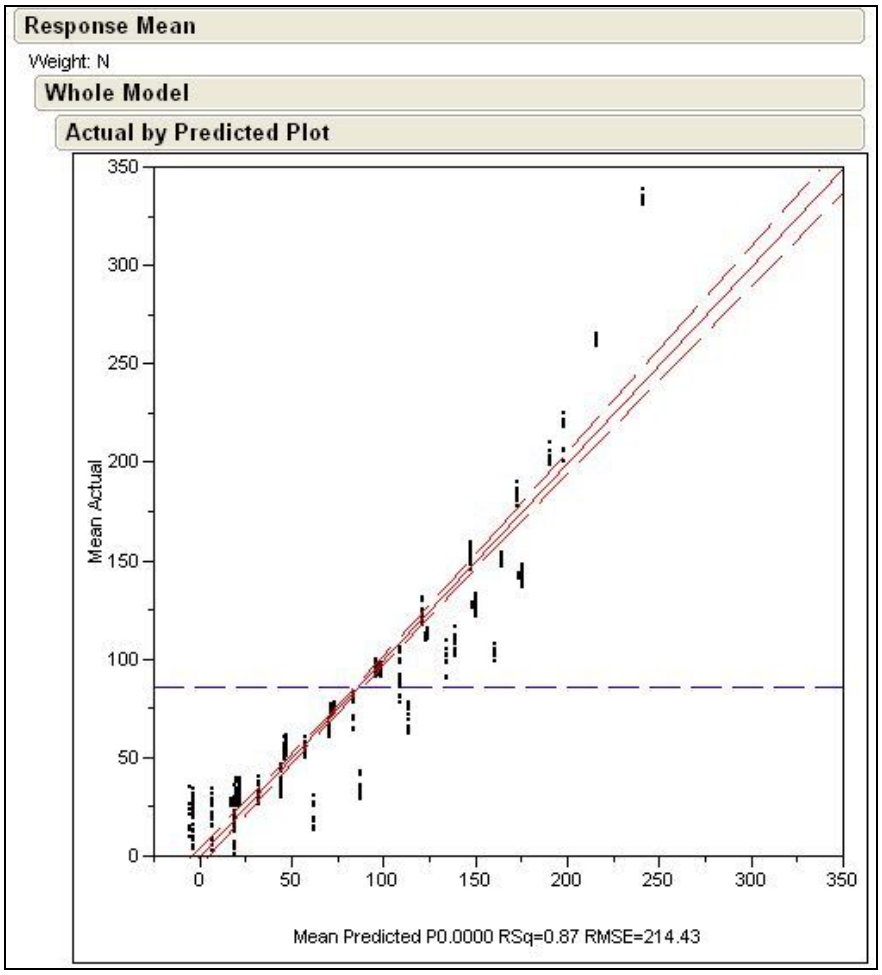


Figure 20. Plot of Predicted vs. Actual Mean Offset for DR Linear Model with Main Effects

Now consider the model which incorporates interactions between the main factors displayed as well as a quadratic effect with current speed presented in Table 9. This model results from performing a stepwise search with refinement and is the best fitting linear model for the Dead Reckoning data set.

Response Mean

Weight: N

Whole Model

Summary of Fit

RSquare	0.982183
RSquare Adj	0.981982
Root Mean Square Error	80.20283
Mean of Response	85.94858
Observations (or Sum Wgts)	63014

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	8	251414553	31426819	4885.635
Error	709	4560638	6432.4938	Prob > F
C. Total	717	255975191		0.0000

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	25.646147	0.735502	34.87	<.0001
currentSpeed	85.421017	0.60449	141.31	0.0000
CurrentOffset{0&45&90&135-180}	-20.61902	0.470433	-43.83	<.0001
CurrentOffset{0&45-90&135}	-6.475071	0.394173	-16.43	<.0001
CurrentOffset{90-135}	-9.571541	0.597129	-16.03	<.0001
(currentSpeed-0.81777)*(CurrentOffset{0&45&90&135-180}-0.68394)	-40.5298	0.735877	-55.08	<.0001
(currentSpeed-0.81777)*(CurrentOffset{0&45-90&135}-0.06281)	-11.1746	0.776926	-14.38	<.0001
(currentSpeed-0.81777)*(CurrentOffset{90-135}+0.02296)	-13.50469	1.164903	-11.59	<.0001
(currentSpeed-0.81777)*(currentSpeed-0.81777)	10.009439	1.037479	9.65	<.0001

Table 9. DR Linear Model Output Predicting Mean Location Offset w/ Interactions

The adjusted R-squared of 0.98 is an improvement over the simple model and all coefficients are statistically significant. The coefficients for the current offset are less easily described in this model because of the complicated interaction effects and the quadratic speed effect. This model more accurately predicts the error than the first, more simple, version, as demonstrated in Figure 21. A small amount of underestimation is still evident at the extremes of the data set, but overall the fit is good. Figure 22 illustrates the predicted location offset as a function of current speed and current offset for the DR model.

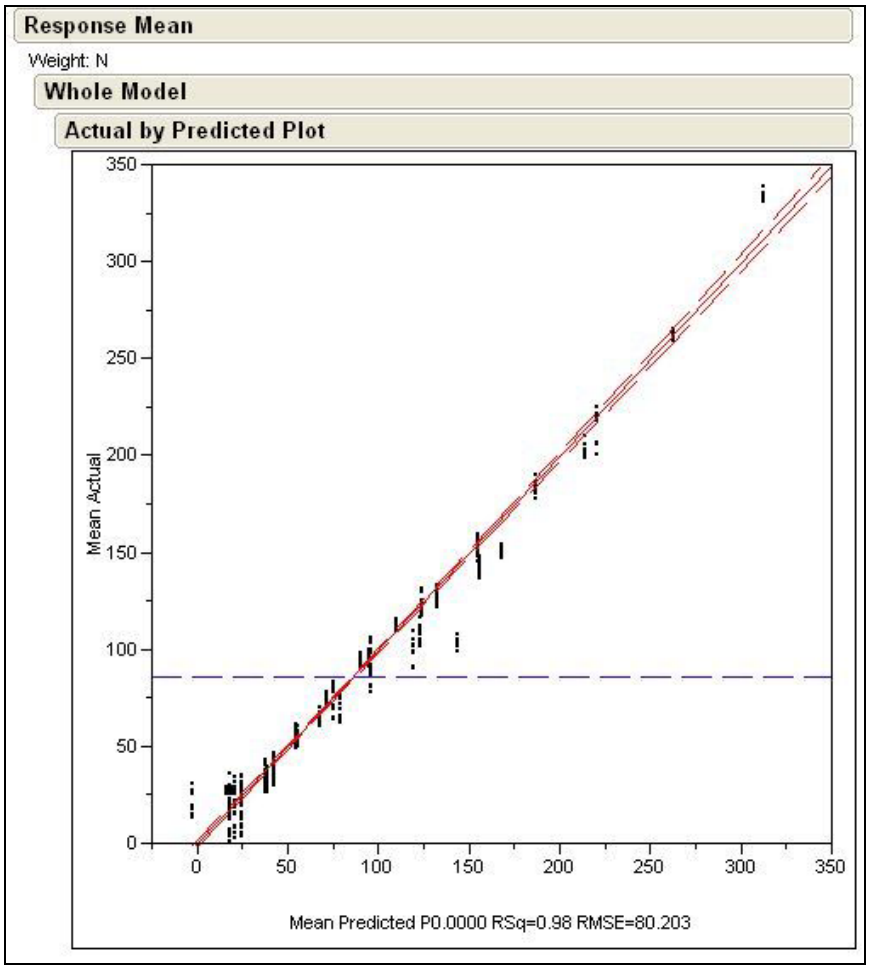


Figure 21. Plot of Predicted vs. Actual Mean Offset for DR Linear Model with Interactions

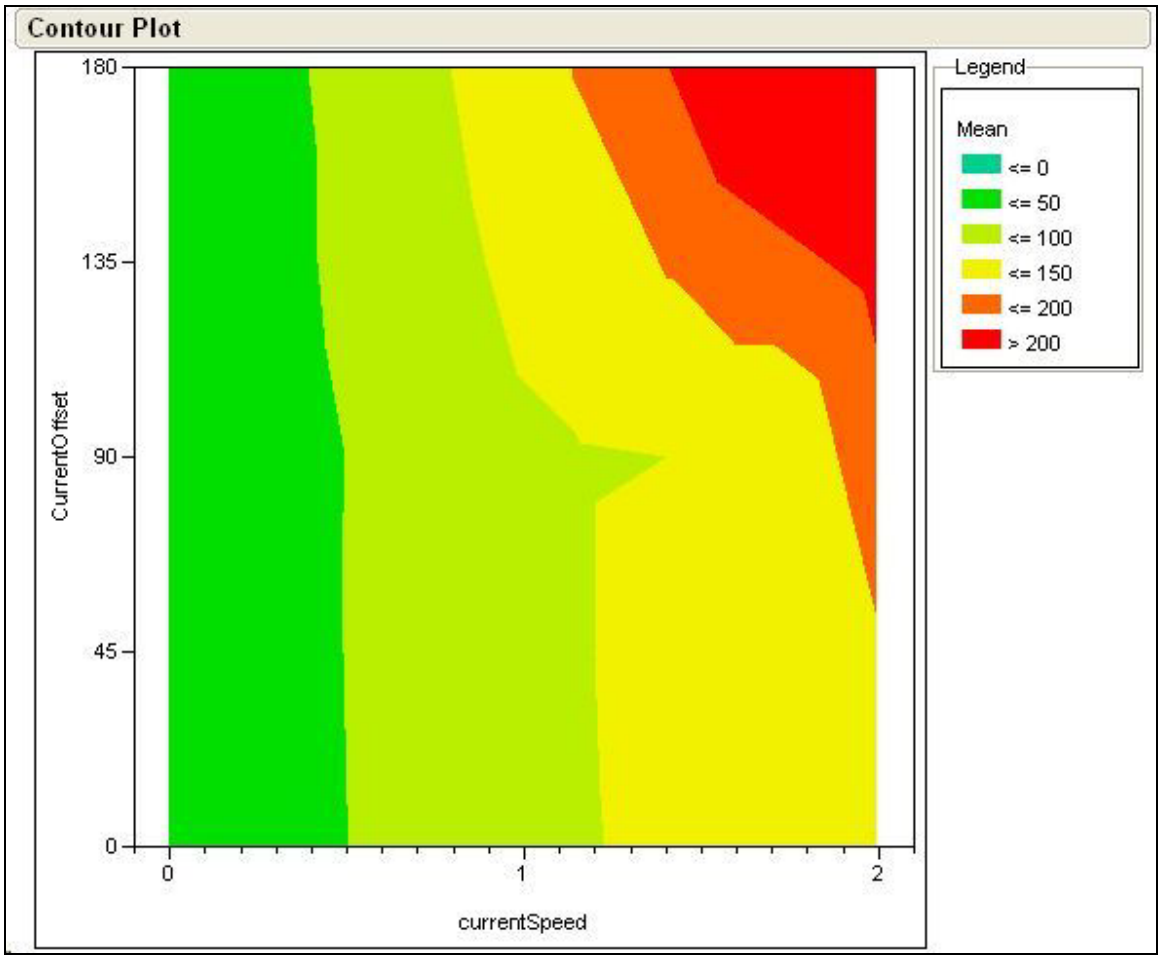


Figure 22. Predicted Mean Offset given Current Speed, Current Offset for the DR Model[5]

Predicted offset remains low at any current direction when the current speed is kept below 0.5 knots. As current speed increases it becomes more important to control the direction of current flow. One important consideration when interpreting Figure 22 is that the offsets in the data set were artificially truncated at 115 feet when current direction was perpendicular to the vehicle's track due to the sensor's range. This model does answer the key question raised by the logit model; namely, which direction of current flow is best for reducing predicted error offset values, zero or 180 degrees? Observe that the gradient at zero degrees is modest in terms of slope as current speed rises. The gradient at 180 degrees is much steeper and the predicted offsets can be very large for current speeds as low as one knot.

[5] This contour plot was derived using predictions from the applicable model.

3. Conclusions for the Dead Reckoning Model

Often the decision to operate the vehicle in the dead reckoning mode is not left to the operator. As previously mentioned the vehicle operates in the DR mode between fixes and the interval between fixes is random because of the uncertainty involved in interpreting the return signal from the transponder. Every effort should be made to operate the vehicle in low current speed conditions. Increasing current speed is the single biggest contributor to error offset in obstacle location. If the vehicle must be operated in moderate to high current conditions then attempts should be made to operate with the current pushing the vehicle from behind. Less time elapses when current is pushing from behind before the obstacle is detected and thus less error will develop in predicted obstacle location.

D. TRANSDUCER MODEL INPUT/OUPUT ANALYSIS

LBL is the most common method of vehicle navigation. The vehicle still relies on Dead Reckoning navigation between fixes, thus the conclusions and observations from the previous section are still applicable. Data generated by the model are used to build a logit model to predict the probability of detection given a combination of input factors and to build a linear regression model to predict the mean location error offset given that detection has occurred. Inputs to the model are considered first, followed by output data format and interpretation. The logit model which predicts detection probability is presented next, followed by a linear regression model which predicts mean offset error. Finally, some observations and conclusions on the transducer model are presented to close this section out.

1. Inputs

Implementation of the four factors from the dead reckoning model, namely current speed, current direction, current noise and maximum compass error, remains unchanged in the transducer model. New factors are discussed here.

a. Transducer Drop Error

Transducer drop error is dependent on GPS accuracy and operator proficiency. The operator positions the transducer over the drop zone using a GPS receiver. Errors develop due to bias and operator error and can reach values of approximately 3.5 meters, or 12 feet. According to recent data collected and analyzed by

the SPACEAF GPS Support Center, 95% of reported GPS positions are within 3 meters of actual. The median value is approximately 1.25 meters [Ref]. These data suggest that using a uniform density distribution may not be appropriate for modeling the drop error. A bivariate normal distribution, with equal X and Y standard deviation, models drop error more accurately and is a reasonable simplification of the actual distribution.

The error is produced in the executable *RemusRunTransducer* class and the transducer offset is set at the beginning of each repetition. The drop error factor is varied by adjusting the standard deviation of the variate to produce a distribution with different spreads. The mean value of the error is considered in eight different levels from a minimum of 3 feet to a maximum of 12 feet. The minimum value corresponds to the best possible accuracy with no operator error. The maximum value takes into account some operator error in placement of the transducers.

b. Current Effect on Transducer Position

The *Transducer* components are passed a *RandomVariate* object used to generate the errors attributed to the effect of current and wave action on the transducer. Current tends to displace the transducer off its ideal location in the direction of current flow. The direction of current flow is accounted for by rotating the bivariate normal distribution so that the major axis (in this case the X-axis) is aligned with current direction. The current speed is accounted for by adjusting the mean of the X component of the variate. A current speed of zero implies a mean of zero and as current speed increases the mean of the X component increases in the positive direction. Current direction and speed factors remain unchanged from the dead reckoning model.

c. Wave Action on Transducers

The same bivariate normal object used for implementation of current effects on the transducer is also used to implement wave action effects. Wave action is considered as a function of sea state. Higher sea states tend to generate larger waves, and larger waves tend to displace the transducer from its ideal location with more forcefulness. Sea state is considered as a factor at five levels ranging from zero to four. Table 10 summarizes sea state number and related wave heights of interest for this model. The vehicle is not operated in sea states exceeding four. Increasing wave action is implemented by increasing the standard deviation of the X component of the bivariate

normal used for current effects. This produces more variance in the reported location of the *Transducer* component and corresponds to a larger ellipse of uncertainty for the component's real location.

Sea State	Description	Wind Description	Wind Velocity (kts)	Average Wave Height (ft)
0	Sea like a mirror.	Calm	0	0
0	Ripples with the appearance of scales are formed, but without foam crests.	Light Air	2	0.05
1	Small wavelets still short but more pronounced; crests have a glassy appearance but do not break.	Light Breeze	5	0.18
2	Large wavelets, crests begin to break. Foam of glassy appearance, perhaps scattered whitecaps.	Gentle Breeze	8.5 – 10	0.6 – 0.88
3	Small waves, becoming longer; fairly frequent whitecaps.	Moderate Breeze	12 – 16	1.4 – 2.9
4	Moderate waves, taking a more pronounced long form; many whitecaps are formed. Chance of some spray.	Fresh Breeze	18 – 20	3.8 – 5.0

Table 10. Sea State Table (from Ocean Technology Systems)

d. Ping Interval

Ping interval is passed to the *RemusMover* at the beginning of each run. Nominally a five second ping interval is utilized by operators because of a perceived tradeoff between navigation accuracy and computing time onboard the vehicle. It may be interesting to determine if reducing the interval improves accuracy significantly, or if increasing the interval degrades accuracy significantly. Ping interval is considered at eight levels between 2 seconds and 9 seconds.

e Other Parameters

Sensor detection range remains set at 115 feet. The vehicle will start from $(x_o,y_o) = (0,0)$ with orders to move to $(x_f,y_f) = (1000,0)$. All dimensions are in feet. The vehicle will travel at a speed of 5 knots. An obstacle is placed at $(x_m,y_m) = (500,0)$. Two

Transducer objects are used for this model. The first one is placed at (x_{t1},y_{t1}) = (0,-100) and the second is placed at (x_{t2},y_{t2}) = (1000,-100).

f. Design Points

Table 11 provides the first eight design points for the transducer model. 50 matrices were produced and combined for a total of 400 design points. The cutoff for pairwise correlation for each individual matrix was 0.40. The cutoff is bigger than that used in the DR model because more factors exist in the transducer model, and the increased complexity meant that in order to keep the total run times manageable fewer matrices were combined.

Current Direction	Current Speed	Current Noise	Max Compass Error	Drop Error	Ping Interval	Sea State
315	1.1429	0	0.0043	4.2857	7	1
45	1.7143	0.1786	0.03	5.5714	4	2
0	1.4286	0.0714	0.0086	10.7143	9	4
180	0.5714	0.1429	0	8.1429	3	3
225	2	0.1071	0.0171	12	2	0
135	0.8571	0.2143	0.0129	3	8	1
270	0	0.25	0.0214	9.4286	5	3
90	0.2857	0.0357	0.0257	6.8571	6	0

Table 11. First Eight Design Points for Transducer Model

Pairwise correlations between columns in the design matrix are presented in Table 12. Note that the combination of multiple smaller matrices results in a maximum pair-wise correlation in the design matrix of less than 0.1.

	Current Direction	*Current Speed*	*Current Noise*	*Max Compass Error*	*Drop Error*	*Ping Interval*	*Sea State*
Current Direction	1						
Current Speed	-0.0838	1					
Current Noise	-0.0072	-0.0038	1				
Max Compass Error	0.0203	0.0387	-0.0504	1			
Drop Error	-0.0010	0.0281	0.0176	0.0053	1		
Ping Interval	-0.0290	0.0229	0.0167	-0.0085	-0.0390	1	
Sea State	0.0452	-0.0452	0.0460	-0.0116	0.0067	0.0059	1

Table 12. Design Matrix Pairwise Correlations for Transducer Model

2. Output Analysis

Output from the transducer model is similar to the output from the dead reckoning model with the addition of levels for the three additional factors. 50 repetitions were conducted at each design point for a total of 20,000 observations.

a. Probability of Detection Given Current Direction, Speed and Sea State

Like the dead reckoning model there are combinations of current speed and direction that always produce detection regardless of the level of any other factor. Table 13 summarizes these points for the transducer model.

	No Detect								
		Current Speed							
		0	0.2857	0.5714	0.8571	1.1429	1.4286	1.7143	2
Current Direction	0	0	0	0	0	0	0	0	0
	45	0	0	0	0	1	2	16	7
	90	0	0	0	3	24	84	110	197
	135	0	0	0	0	16	11	96	366
	180	0	0	0	0	0	0	0	0
	225	0	0	0	2	2	18	76	171
	270	0	0	0	1	18	122	227	342
	315	0	0	0	0	1	5	0	14
	Detect								
		Current Speed							
		0	0.2857	0.5714	0.8571	1.1429	1.4286	1.7143	2
Current Direction	0	200	250	150	300	350	300	650	300
	45	200	400	300	200	349	248	434	343
	90	500	350	500	147	376	166	40	3
	135	150	350	200	450	334	139	254	134
	180	400	250	350	300	300	250	350	300
	225	250	300	400	498	248	332	124	79
	270	400	400	250	149	132	378	73	8
	315	400	200	350	450	349	445	50	236

Table 13. Current Direction, Current Speed vs. Detect for Transducer Model

Current speeds less than 0.8571 knots produce detection regardless of the level of any other factor, including current direction. Current directions of 0 and 180 degrees also produce detection regardless of other factors. Observations with these traits are removed from the data set without adverse impact on the model. The conclusion is that current speeds of less than 0.8571 knots or current directions of 0 or 180 degrees

produce detection with probability 1. Recall that current directions are aggregated by relative direction with respect to the vehicle's motion, as in the dead reckoning model.

The logit model again provides prediction of probability given some combination of input factors. The logit is regressed on the 7 factors and the resulting model is provided in Table 14. Current offset is treated as a categorical variable in the model. The remaining factors are treated as continuous variables. Sea state could be considered as a categorical variable since there is no such thing as a sea state 1.5, for example, but simplicity is gained by treating it as a continuous variable.

Ordinal Logistic Fit for Detection

Whole Model Test

Model	-LogLikelihood	DF	ChiSquare	Prob>ChiSq
Difference	2581.4236	7	5162.847	0.0000
Full	2123.2599			
Reduced	4704.6835			

RSquare (U)	0.5487
Observations (or Sum Wgts)	9100

Converged by Gradient

Parameter Estimates

Term	Estimate	Std Error	ChiSquare	Prob>ChiSq
Intercept[1]	-14.309342	0.3798061	1419.4	<.0001
Current Speed	6.29847436	0.1735105	1317.7	<.0001
Current Noise	0.94771228	0.4470515	4.49	0.0340
Max Compass Error	10.0485057	3.9779596	6.38	0.0115
Mean Drop Error	0.03143732	0.0146921	4.58	0.0324
Ping Interval	0.14088641	0.0174223	65.39	<.0001
Current Offset{45-90&135}	-2.3125808	0.0828843	778.48	<.0001
Current Offset{90-135}	0.96813221	0.0477581	410.94	<.0001

Table 14. Logit Main Effect Model for Transducer Run

The first observation is that not all of the factors are statistically significant in this model. Sea state had a p-value greater than 0.05 and was thus dismissed from the model. This suggests, at least in this fairly straightforward model, that sea state did not play a significant role in predicting the probability of detection of an obstacle. The base current direction, given by the intercept term, is when current offset is 135 degrees relative to the vehicle. The coefficients make sense in this model. The probability of detection should decrease as current speed increases and the positive coefficient for

current speed supports this expectation. The probability of detection should decrease as the interval between fixes increase and the positive coefficient for ping interval supports this expectation.

The model has an R-squared(U) value of 0.55 indicating that the model does not explain a tremendous amount of uncertainty. This observation is not surprising given the relative simplicity of the main effect logit model. Table 15 presents the final logit model for the transducer model. This model results from performing a stepwise search with refinement and is the best fitting logit model for the transducer model data set.

Ordinal Logistic Fit for Detection

Whole Model Test

Model	-LogLikelihood	DF	ChiSquare	Prob>ChiSq
Difference	2757.9313	14	5515.863	0.0000
Full	1946.7521			
Reduced	4704.6835			

RSquare (U)	0.5862
Observations (or Sum Wgts)	9100

Converged by Objective

Parameter Estimates

Term	Estimate	Std Error	ChiSquare	Prob>ChiSq
Intercept[1]	-19.246234	0.6125213	987.30	<.0001
Current Speed	7.41687337	0.2319813	1022.2	<.0001
Current Noise	0.45093055	0.4800818	0.88	0.3476
Max Compass Error	172.371928	11.082689	241.90	<.0001
(Current Speed-1.41759)*(Max Compass Error-0.01521)	-307.51962	20.659722	221.56	<.0001
Mean Drop Error	-0.0024747	0.015277	0.03	0.8713
(Current Noise-0.12205)*(Mean Drop Error-7.64835)	-0.4952277	0.1701987	8.47	0.0036
Ping Interval	0.16851683	0.0222195	57.52	<.0001
(Max Compass Error-0.01521)*(Ping Interval-5.46703)	-12.25082	2.3607478	26.93	<.0001
Sea State	0.0428638	0.0336428	1.62	0.2026
(Ping Interval-5.46703)*(Sea State-1.90659)	0.04632603	0.0158172	8.58	0.0034
Current Offset{45-90&135}	-2.8954708	0.1347084	462.01	<.0001
Current Offset{90-135}	1.09943748	0.0559851	385.65	<.0001
(Max Compass Error-0.01521)*(Current Offset{45-90&135}+0.30769)	128.653033	11.73858	120.12	<.0001
(Max Compass Error-0.01521)*(Current Offset{90-135}+0.08242)	-51.341884	5.5070883	86.92	<.0001

Table 15. Logit Interaction Effect Model for Transducer Run

This model includes the sea state, current noise and mean drop error factors even though the p-values are greater than 0.05 because the factors are included in interactions with other factors. All of the coefficients make sense in as much as they can be easily interpreted. The R-squared(U) has improved to 0.59, a small increase that implies a little more uncertainty is explained by this more complicated model. It should

be noted that while the more complex model involving interactions explains more uncertainty it does so at the expense of ease of use. The simpler model using only main effects may be more useful to an operator due to the ease of use. Figure 23 illustrates the probability of detection given a combination of current speed and offset.

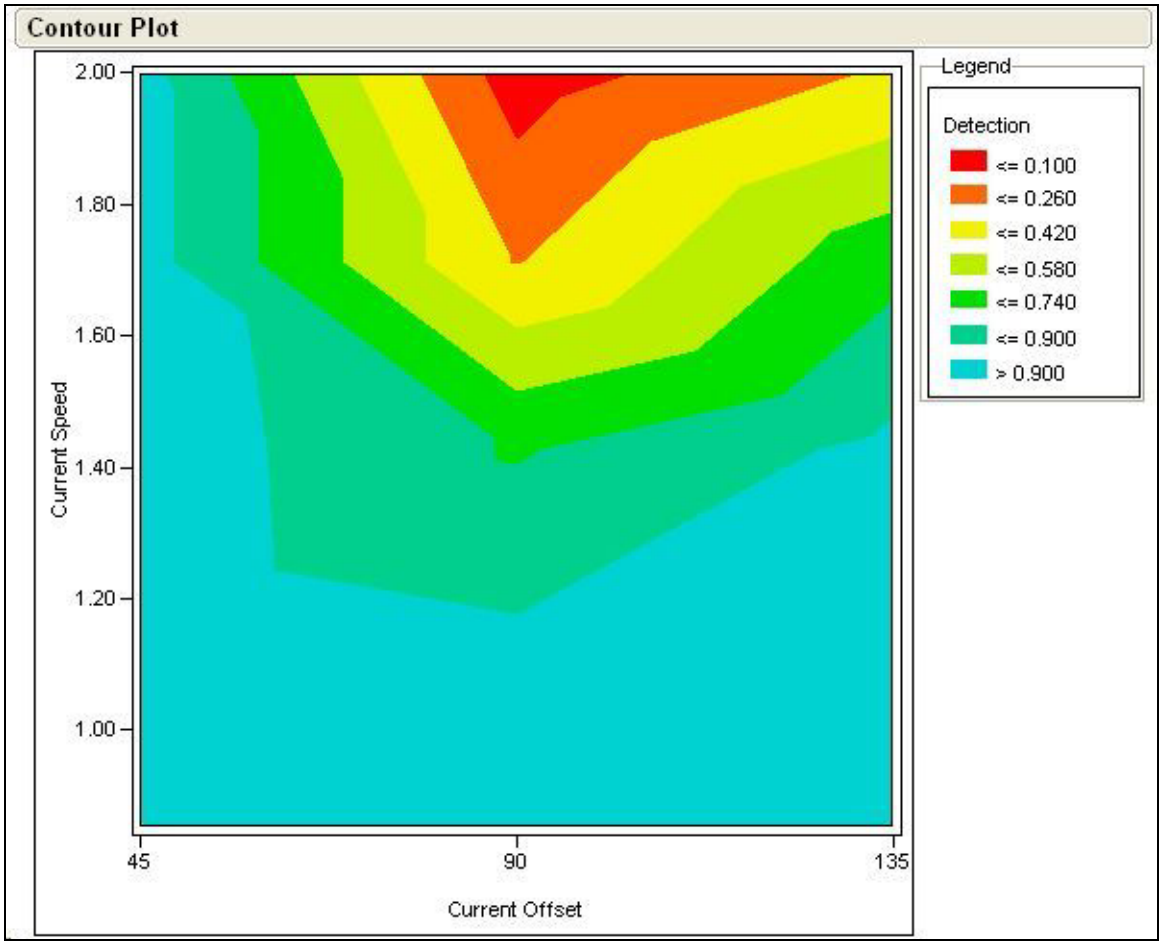


Figure 23. Predicted Probability of Detection Given Current Speed, Offset for Transducer Model[6]

As expected, the steepest gradient occurs when the current offset is 90 degrees, or directly across the vehicle's path. Recall that the probability of detection is 1 when current offset is either 0 or 180 degrees, and when current speed is less than approximately 0.85 knots. These numbers represent an improvement over the dead reckoning model. Higher current speeds are alright in the transducer model and a current offset of 180 degrees results in detection with probability 1. Clearly the vehicle should be

6 This contour plot was derived using predictions from the applicable model.

operated with current either opposing or aiding the vehicle path. Once again the question about which direction is best is addressed more effectively by modeling the mean offset as a function of the input factors which the operator has control over.

b. Magnitude of Error Given Detection Occurs

The mean offset is modeled under the assumption that detection has occurred, thus all entries in the transducer data set for non-detection are eliminated prior to building the model.

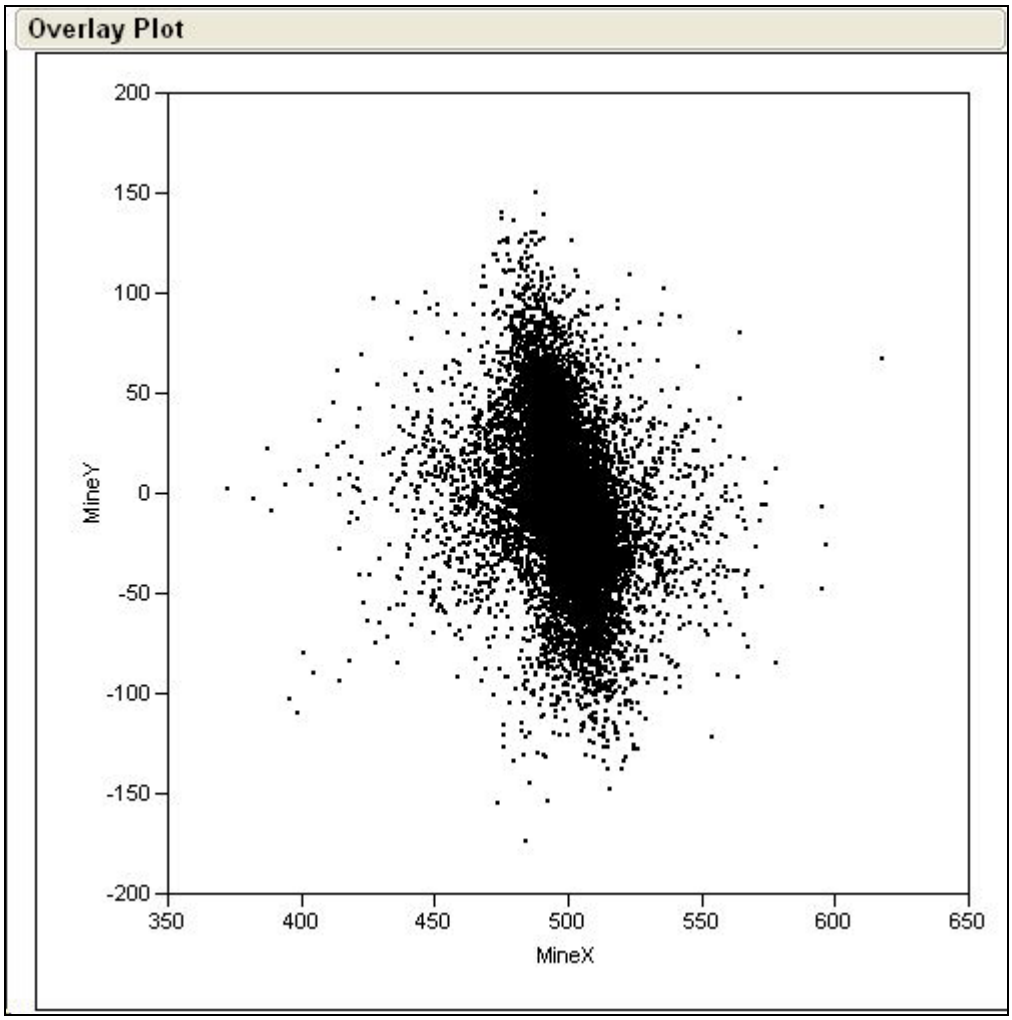


Figure 24. Plot of MineX and MineY Locations for Transducer Model

Figure 24 reveals a density that is more centrally located than in the dead reckoning model. This suggests that accuracy in predicting location is improved in the transducer model. The distribution is still roughly symmetric around the y axis and is now roughly symmetric around the x axis. Table 16 illustrates this more clearly.

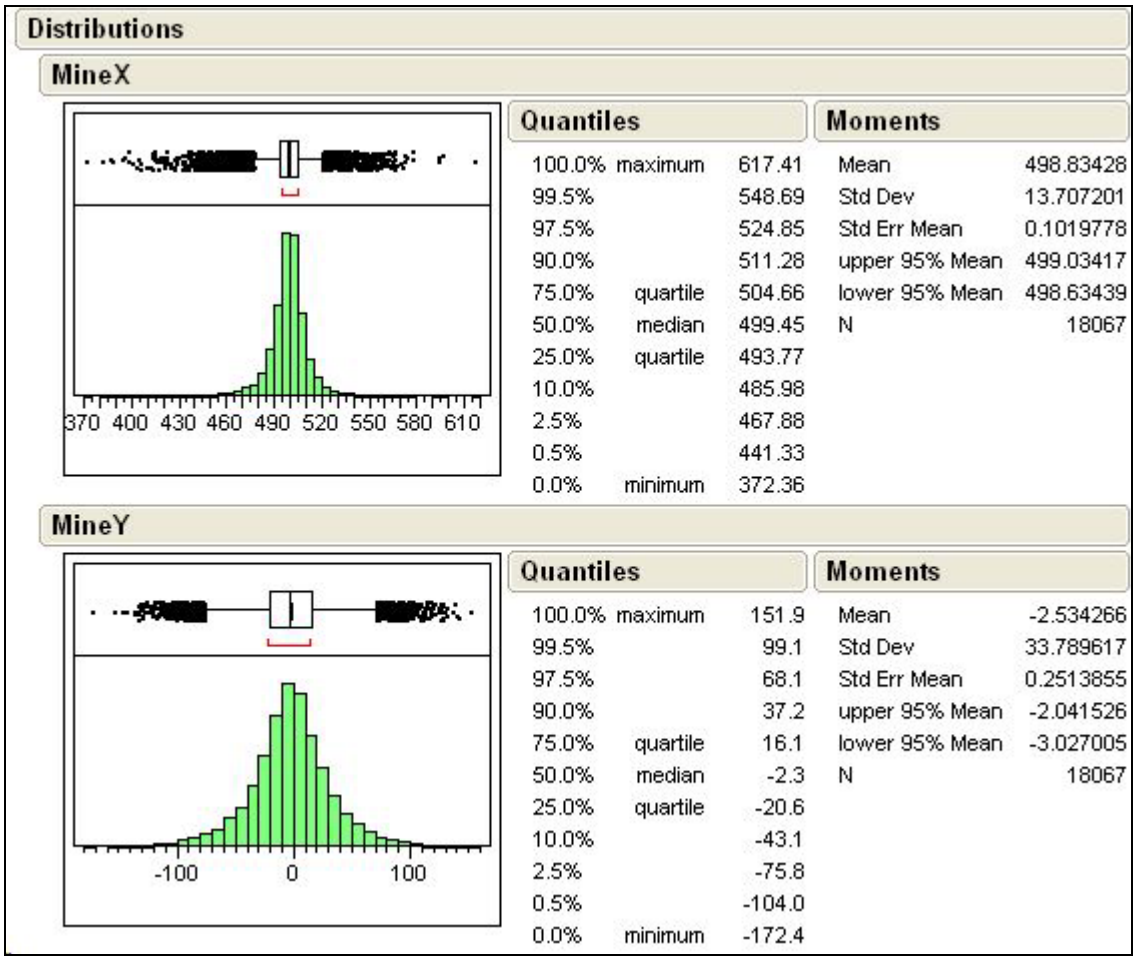


Table 16. Summary Statistics for Transducer Run MineX, MineY

The key distinction to make when comparing Table 16 to Table 7 from the dead reckoning model is that both distributions have a lower proportion of points in the tails implying greater accuracy in predicting location. In particular, note the improvement in the 25% and 75% quantiles for the transducer model. In all instances there is a marked improvement in the value for the transducer model. Recall that the mine location was set at $(x_m, y_m) = (500, 0)$.

The operator has control over five of the seven factors. Current speed and offset are controlled by choosing when and where to operate the vehicle. Sea state may be similarly controlled, but only if the operator chooses not to operate the vehicle in high sea states. Transducer drop error may be controlled by attention to detail. Ping interval is controlled in advance by software implementation. The first model predicting location

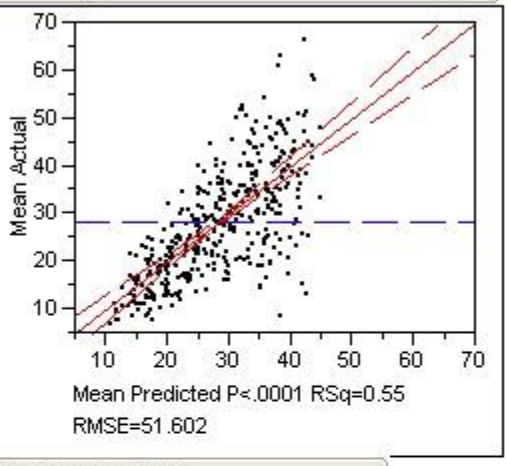
offset models the mine offset against current speed, current offset, mean drop error, ping interval and sea state main effects. Refer to Table 17 for model results.

Response Mean

Weight: N

Whole Model

Actual by Predicted Plot



Summary of Fit

RSquare	0.546419
RSquare Adj	0.540513
Root Mean Square Error	51.60236
Mean of Response	28.20602
Observations (or Sum Wgts)	18026

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	5	1231804.2	246361	92.5193
Error	384	1022516.7	2663	Prob > F
C. Total	389	2254320.8		<.0001

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.4570673	1.620718	0.28	0.7781
Current Speed	4.4064758	0.628239	7.01	<.0001
Mean Drop Error	2.5002395	0.129776	19.27	<.0001
Ping Interval	0.5160673	0.167834	3.07	0.0023
Sea State	1.0807349	0.275359	3.92	0.0001
Current Offset{45-90}	1.8325997	0.561835	3.26	0.0012

Table 17. Transducer Linear Main Effect Model Predicting Location Offset

Current offset is treated as a categorical variable in this model. Current offsets of 0, 90 and 135 degrees were not statistically significant and are omitted. The base case of current offset is 180 degrees and is represented by the intercept. All of the coefficients make sense. A higher current speed or larger drop error will increase the

location error. The positive coefficient for ping interval indicates that an increase in ping interval causes an increase in prediction error. The adjusted R-squared value of 0.54 indicates that not a lot of variance is explained by the model. The plot of predicted vs. actual offsets in Table 17 exhibits signs of heteroscedasticity since the variance is larger when the predicted offset is high.

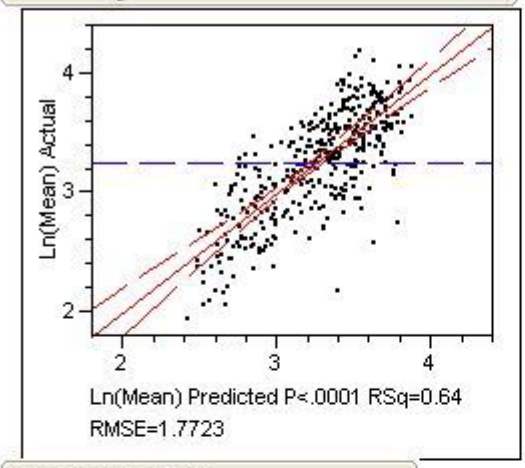
Now consider the offset prediction model incorporating interactions between main factors presented in Table 18. This model results from performing a stepwise search with refinement and is the best fitting linear model for the transducer data set. The adjusted R-squared has improved to 0.64 indicating that more variability is explained by this model. The residuals fit the prediction better than the main effect model in that there is no discernable pattern to them. All of the coefficients make sense. The most influential factor, other than current speed, is mean drop error.

Response Ln(Mean)

Weight: N

Whole Model

Actual by Predicted Plot



Summary of Fit

RSquare	0.644285
RSquare Adj	0.63586
Root Mean Square Error	1.772263
Mean of Response	3.253172
Observations (or Sum Wgts)	18026

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	9	2161.8013	240.200	76.4746
Error	380	1193.5478	3.141	Prob > F
C. Total	389	3355.3491		<.0001

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	2.1798242	0.05621	38.78	<.0001
Current Speed	0.1518549	0.021711	6.99	<.0001
Mean Drop Error	0.0959369	0.004487	21.38	<.0001
(Current Speed-0.90983)*(Mean Drop Error-7.45813)	-0.033871	0.007627	-4.44	<.0001
Ping Interval	0.0259758	0.005811	4.47	<.0001
(Current Speed-0.90983)*(Ping Interval-5.4387)	0.0489803	0.009864	4.97	<.0001
(Mean Drop Error-7.45813)*(Ping Interval-5.4387)	-0.007685	0.001919	-4.01	<.0001
Sea State	0.0435198	0.009542	4.56	<.0001
Current Offset{45-90&135}	0.0633194	0.016186	3.91	0.0001
Current Offset{90-135}	-0.081921	0.020015	-4.09	<.0001

Table 18. Transducer Linear Interaction Model Predicting Location Offset

A contour plot of predicted offset for a given mean drop error and ping interval is presented in Figure 25. In general, increasing the ping interval tends to

increase the drop error for all drop errors. Larger drop errors tend to increase the predicted offset. This result comes from the integral role the transducers play in vehicle navigation.

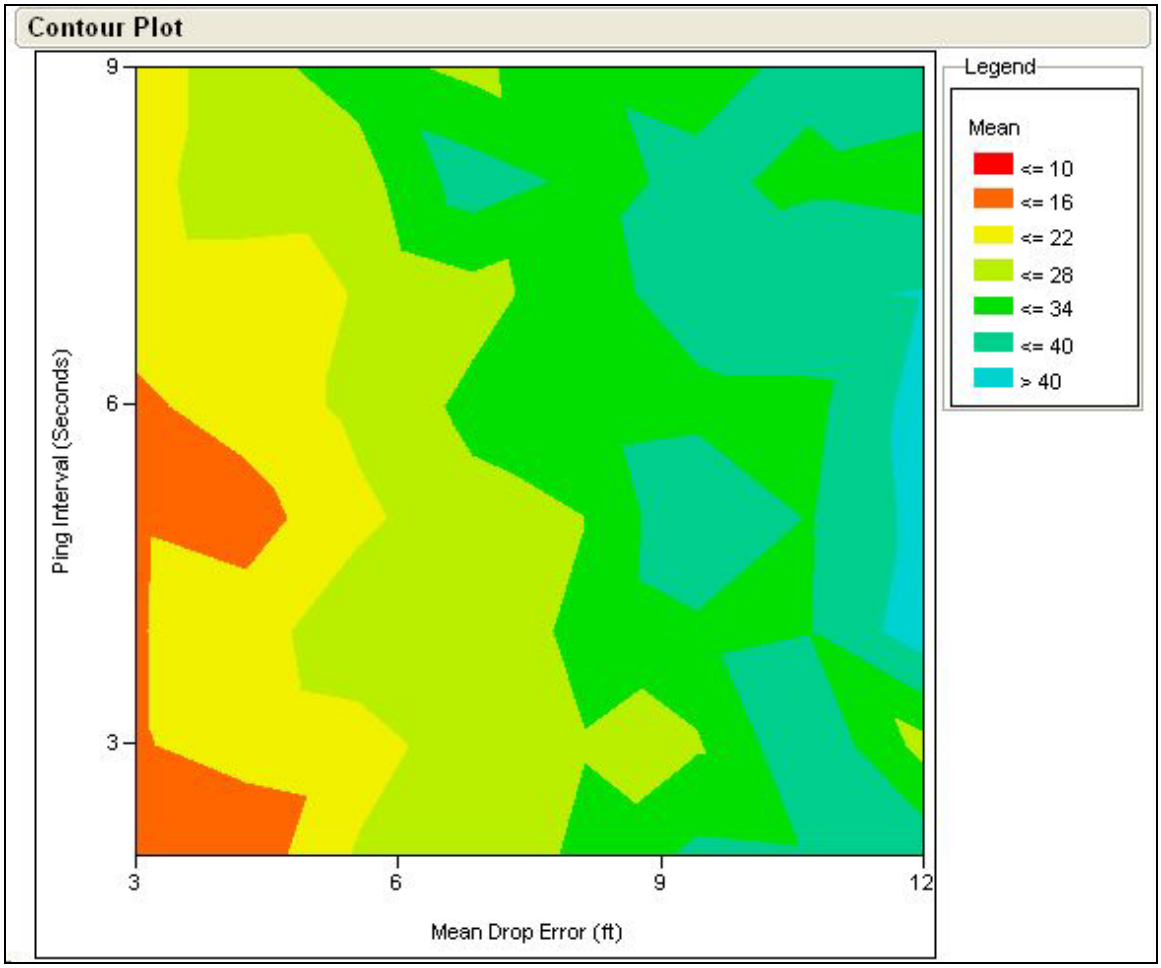


Figure 25. Predicted Offset Given Mean Drop Error, Ping Interval for Transducer Model[7]

[7] This contour plot was derived using predictions from the applicable model.

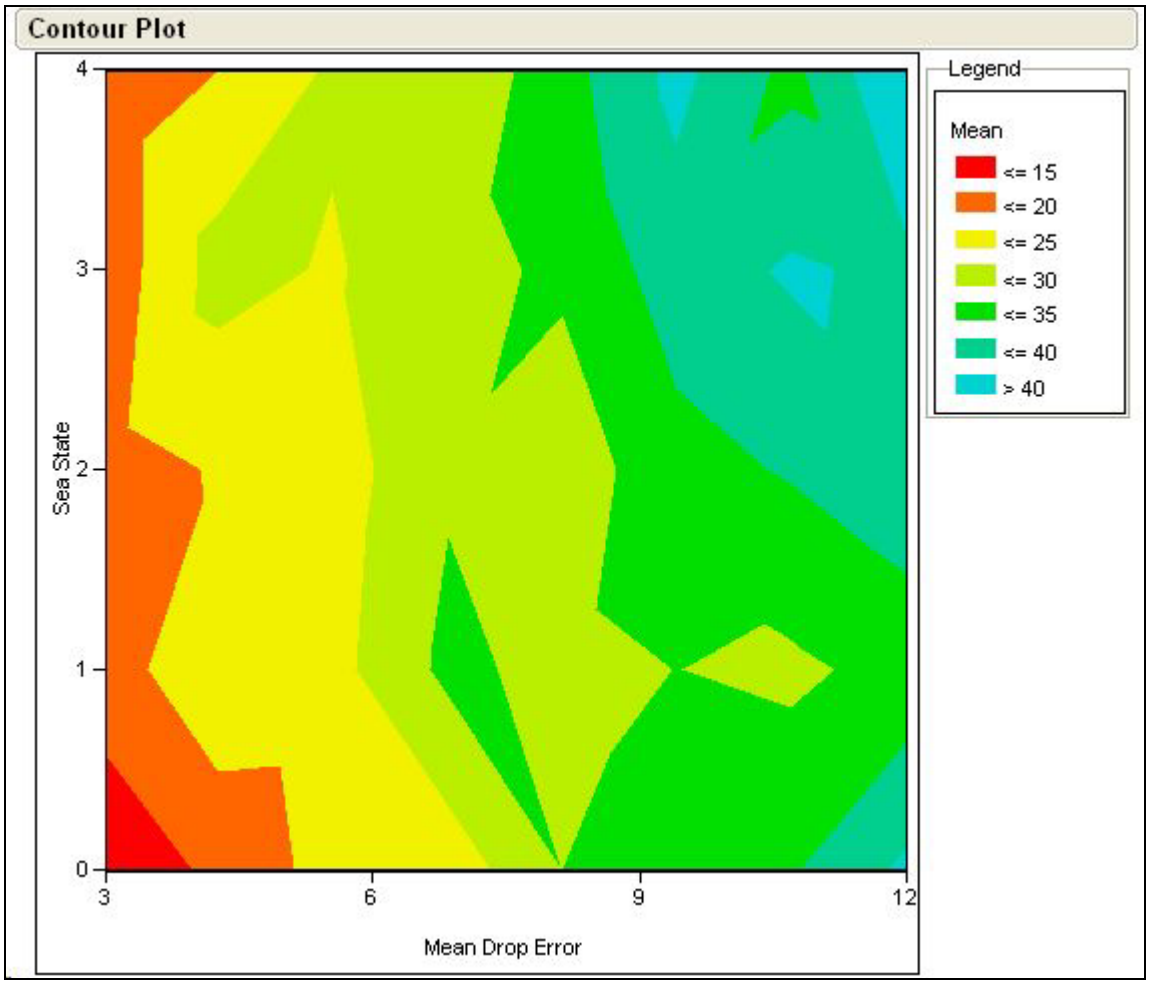


Figure 26. Predicted Offset Given Mean Drop Error, Sea State for Transducer Model[8]

Figure 26 presents predicted offset given mean drop error and sea state and illustrates the effect increasing sea state has on offset prediction. At higher sea states the vehicle has a more difficult time staying on track due to transducer sway and prediction accuracy suffers.

3. Conclusions for the Transducer Model

Operating the REMUS vehicle with transducers is clearly the desired mode of operation. Understanding the impact of factors under the operators control can improve prediction accuracy and thus reduce the amount of time necessary to locate the obstacle post mission. Ideally the vehicle should be operated in low current conditions with very little wave action and the operator should endeavor to place the transducers with maximum accuracy. Ping interval would be set as low as possible determined by the

[8] This contour plot was derived using predictions from the applicable model.

maximum distance the vehicle is expected to operate from the transducers. Realistically these conditions are not always possible.

If the current speed cannot be minimized the vehicle should be operated with the current acting directly from ahead or behind to maximize probability of detection. The maximum errors tend to occur in the plane perpendicular to vehicle motion so the vehicle should not be operated with current acting perpendicular to the vehicle's track. Figures 25 and 26 emphasize the importance of minimizing the mean drop error associated with the transducers. Mean drop error is perhaps the one factor most under the operator's control and pains must be taken to accurately place each one.

E. AREA SWEEP MODEL – "MOWING THE LAWN"

The REMUS vehicle is not expected to be employed in a manner in which it is run across a field of interest one time with the expectation of finding an obstacle. Knowledge of mine location is rarely known beforehand. A more suitable employment of the vehicle is to "mow the lawn". The idea is to make multiple parallel passes through in area to find a randomly located object.

The implementation of this tactic is relatively straightforward with the REMUS simulation model. No modifications are required to the main components and no discussion of model development is required in Chapter III. The major alteration is to develop an executable class with multiple waypoints for the vehicle and a method of randomly placing mines in a field. Of interest to the operator is the probability that the prediction error is less than some threshold value determined by other factors, such as bottom time for divers or visibility concerns.

1. Model Inputs

a. Factors from the Transducer Model

The seven input factors from the transducer model are utilized for the area sweep model. Sea state, current noise, compass error, ping interval, current speed and drop error are treated at a hypothetical "best" level for the area sweep model. Current direction is treated at varying levels.

Sea state is established at 3 to account for small wave action in shallow water areas where the vehicle is most likely to operate. Current noise is established at 0.2

knots. Maximum compass error is set at 0.03% and ping interval is set at 5 seconds. Current speed is set at one knot and drop error mean is set at 5 feet. All of these levels represent conservative, middle of the road estimates for the parameters. Current direction is considered at eight different levels around the compass, as in the preceding models.

b. Other Parameters

Two transducers are used in the area sweep model. They are placed at (x_t,y_t) = (0, -100) and (1500, -100). These points represent the edges of the field being surveyed and reflect operating considerations currently employed. The vehicle will start at (x_m,y_m) = (0, 0) and survey a path similar to that represented in Figure 27. A mine will be randomly placed inside the field at the beginning of each run following a uniform distribution. Only the first detection is recorded for output. Sensor range remains 115 feet and the vehicle travels at a speed of 5 knots. The survey area is 1500 feet wide by 1000 feet tall. The vehicle enters the survey area at (0, 0).

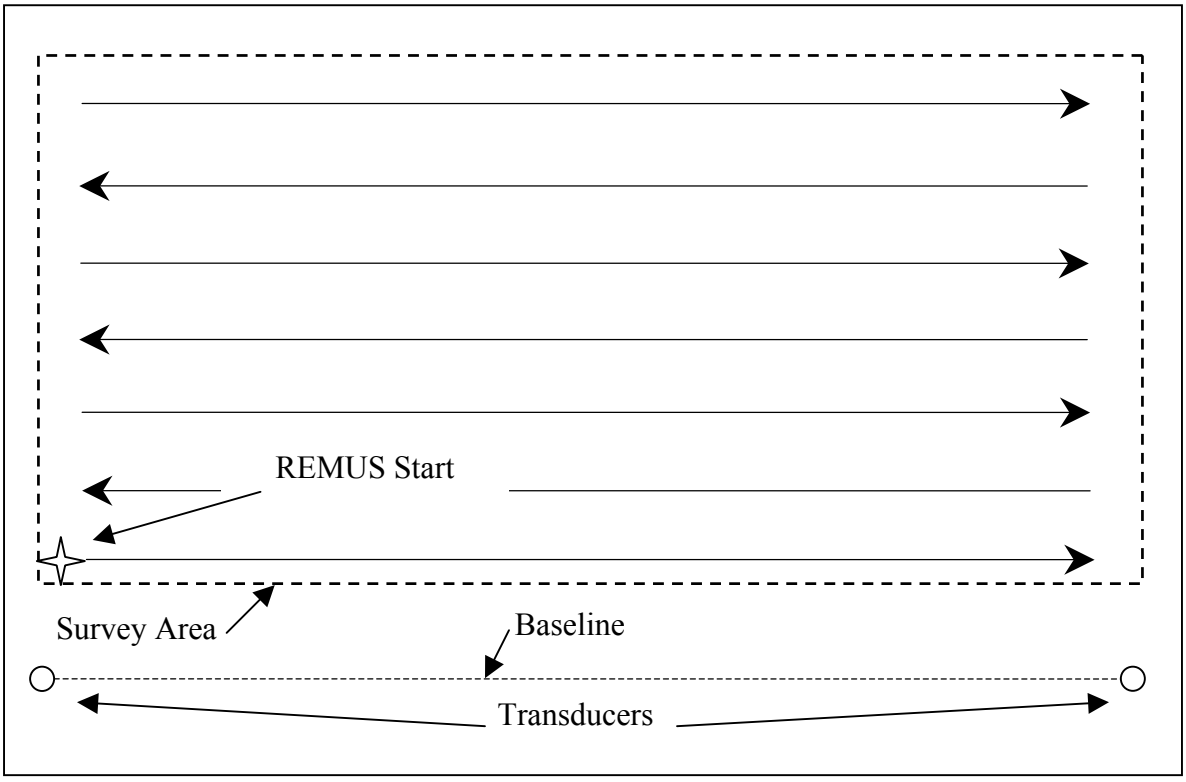


Figure 27. Typical Survey Area Setup

2. Output Analysis

Fifty mines are placed for each current direction. Fifty repetitions are conducted for each mine placement for a total of 20,000 observations. Output data includes the current direction, actual and predicted mine location and a flag denoting whether detection has occurred for each repetition.

The logit model of Table 15 predicts a probability of detection of approximately 0.899 for the given area sweep model parameters. The parameters for the area sweep model were aggregated due to the differing current directions used in this model. The observed probability of detect for an individual run, obtained by calculating the ratio of detections to runs for the data set, is approximately 0.93 with a 95% confidence interval of (0.930, 0.937). The standard deviation of the probability of detection is 0.25 for a single run. The logit model has performed well in predicting the detection probability for this experiment. Figure 28 presents a histogram of the offset predictions given that detection has occurred. The distribution of errors generally follows an Exponential distribution with a mean of approximately 33.6 feet with a 95% confidence interval of (33.1, 34.1) feet. The standard deviation of the errors is approximately 32.9 feet. The mean offset predicted by the transducer model of Table 18 is approximately 32.7 feet. This prediction is also pretty good.

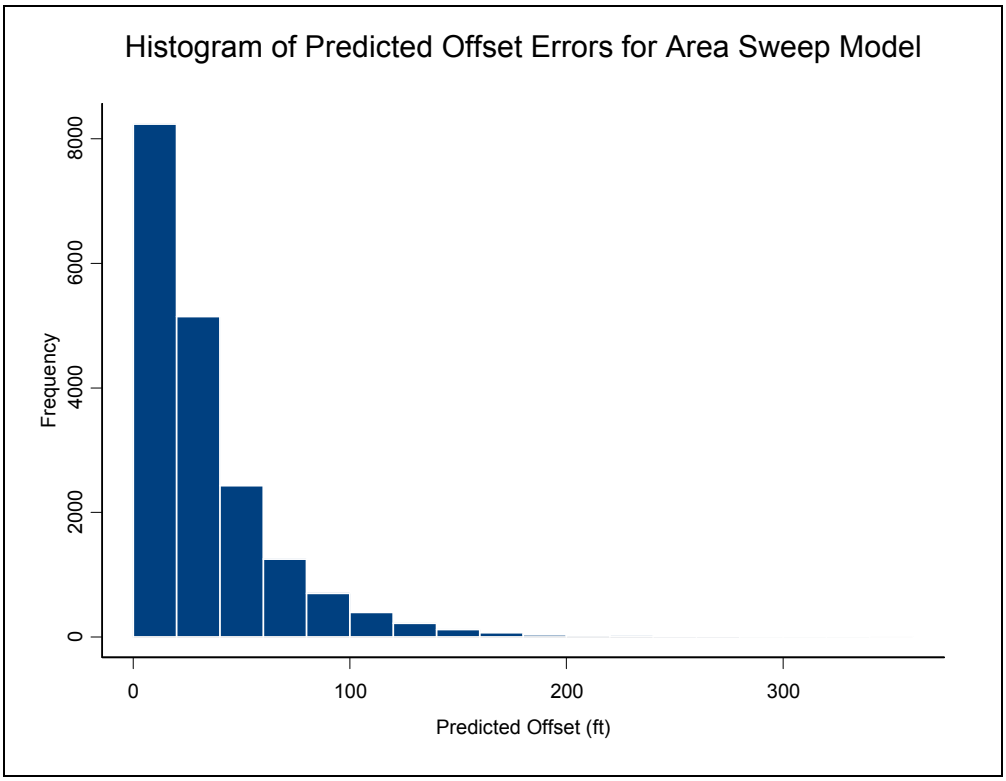


Figure 28. Histogram of Predicted Offset Errors for Area Sweep Model

If the distribution of prediction errors can be established to be exponential with some mean value, then the probability that expected offset is less than some value X could be calculated in a fairly straightforward manner. Consider the QQ plot presented in Figure 29.

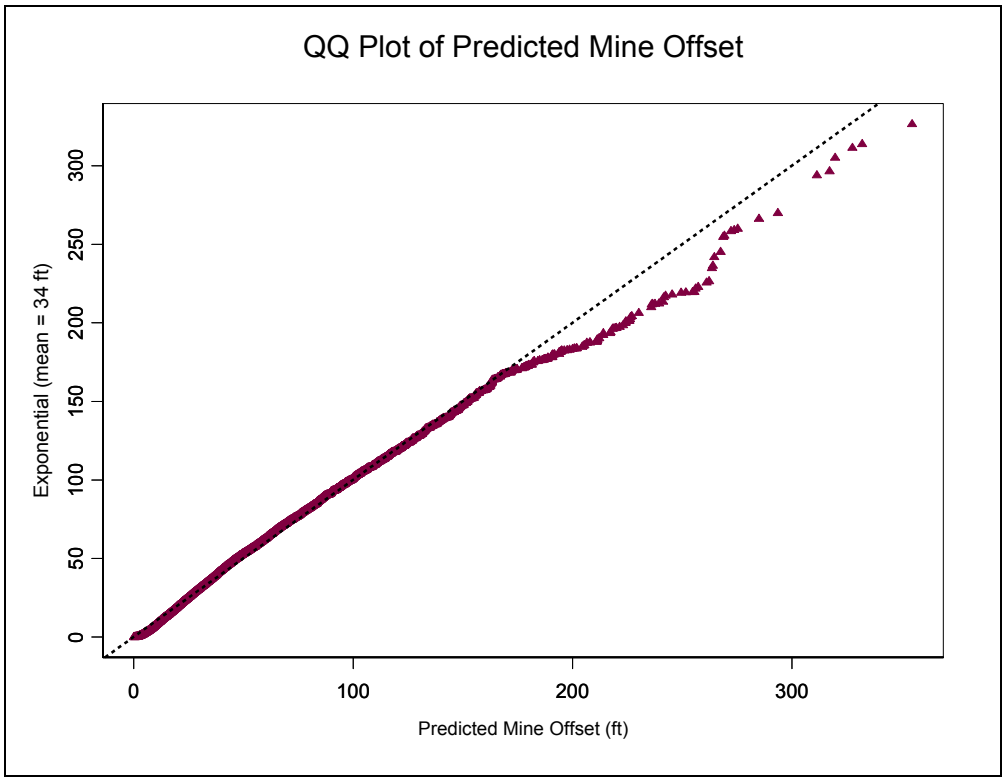


Figure 29. QQ Plot of Predicted Mine Offset

The dashed line has slope one and intercept zero. If the distribution of prediction errors is indeed exponential with mean 33.6 feet then all of the points would lie on the line. This trend is maintained for low and mid offset predictions but lapses somewhat in the tail. However, even in the tail the distribution does not stray too far off the line. Additionally, there are far fewer observations in the tail, as evidenced by the lower density of points in the graph. The claim that prediction errors are distributed exponentially with mean 33.6 feet is supported by the QQ plot of Figure 29 and the histogram of Figure 28.

The implication of the exponential distribution of prediction offset is that determining a probability that prediction offset will be less than some value is easy. The probability that X is less than some value is given by

$$F(x) = 1 - e^{-\lambda x} \text{ for } x \geq 0 \tag{14}$$

where λ equals the inverse of the mean. Additionally, the observed distribution of predicted mine offset has a lighter tail in the larger predicted offsets and thus the

exponential distribution will tend to conservatively predict offset distances. Operators in the field can readily apply this relationship. They can calculate the predicted mean offset given some combination of input factors using the model from Table 18 and then estimate the probability of the detection range given the mean offset. The generated ranges can be used to determine how much time is expected to be needed to search an area to relocate the mine for later clearing.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. ANALYTICAL CONCLUSIONS

Recall that the purpose of the REMUS model is to be an exploratory model that provides insights into vehicle behavior which may be exploited by operators to improve performance. The model provides no exact, correct answer. Accuracy of the output of the model is closely coupled to the accuracy of the input parameter assumptions. Simplifications have been made in the model to allow for easier implementation in code. Improvement of input parameter quality should result in improved predictive ability in the model. Any analyst would be well advised to remember the old adage, "garbage in – garbage out". Any output of this model should therefore be viewed with an eye toward gaining insight and not toward gaining the absolute answer.

There are three steps involved for the operator in attempting to predict the probability of a certain offset value, corresponding to the three flavors of models produced from the REMUS simulation output. First, the operator should be interested in predicting the probability of detection given certain operating conditions. Low current speed and current directions opposing or aiding the vehicle should provide the most favorable conditions for mine location. Current directions that are perpendicular to the vehicle's track should be avoided unless current speed can be maintained below about a half knot or so. Ping interval has a small effect on predicting probabilities. A lower ping interval is better for maximizing probability of detection; however, the ping interval is also determined by the expected operating distance from the transducers. Transducer mean drop error can be controlled to a certain extent by the operator and efforts must be made to minimize the error by accurately placing the transducer at the pre-determined drop point.

Secondly, the operator should predict the mean value of the mine location offset given some combination of factors. Once again, current speed should be minimized in order to minimize expected prediction accuracy. Sea state begins to play a role in predicting accuracy. Current speed, current direction and sea state are variables over which the operator has control only so far as the operator can choose when and where to

deploy the vehicle. It is not hard to imagine situations in which the operator may have to operate the vehicle in less than ideal conditions. Efforts must be made to minimize transducer drop error and the lowest possible value for ping interval should be used to improve accuracy.

Thirdly, once the expected mean offset is calculated the operator can determine the expected probability that the object's actual location is within a certain distance from the predicted location using the exponential distribution CDF. The desired distance may be determined by water conditions, depth or even diver experience and is integral in clearing a minefield.

In general, most of the error in the transducer model accumulates as a result of the vehicle dead reckoning between good fixes. The rest of the error comes from uncertainty associated with triangulation of the vehicle's position. Recommendations for current speed, direction and ping interval are geared towards minimizing the error from dead reckoning. Recommendations for sea state and transducer drop error serve to minimize the triangulation error.

B. RECOMMENDATIONS

Both the Dead Reckoning and Transducer models indicate that operating the vehicle with lower current speed improves both the prediction probability of detecting a mine and the offset error given the mine is detected. If higher current speeds cannot be avoided, the Dead Reckoning model clearly illustrates that operating the vehicle with the current pushing the vehicle either directly from behind or ahead are the most advantageous for improving prediction probability and offset error. Therefore, operate the vehicle with current speeds as low as possible and with current pushing the vehicle from behind to minimize offset errors and maximize detection probability.

The Transducer model indicates that a shorter ping interval is better for improving both prediction accuracy and offset error. A tradeoff is made when shorter ping intervals are selected because maximum allowable range between the vehicle and the transducers decreases as ping interval decreases. Therefore, use as short a ping interval as allowed by operating considerations and minimize transducer drop error to improve accuracy.

The Transducer model indicates that poor performance while placing transducers in the field will result in decreased performance when predicting detection probability and offset errors. Placement error cannot be eliminated due to errors in using GPS systems to locate the drop area, but operators can improve performance by paying attention to detail while positioning the transducers. Therefore, improve transducer placement error by placing a higher priority on accurately locating the drop area for the transducers.

Comparison of results from the Dead Reckoning model and the Transducer model suggests that dead reckoning simply does not provide enough predictive power or accuracy to be relied upon as the primary navigation method for the vehicle. Detection probability improved when transducers were utilized, and experience in the field backs this up. The distribution of predicted offset errors from the Dead Reckoning model had heavier tails, and thus degraded performance, than offset errors from the Transducer model. Therefore, operate the vehicle with transducers in the LBL navigation mode.

The model has produced results that make sense intuitively, but little data exists to validate those results. Therefore, the model may be used to gain insight into the effect of combinations of inputs on prediction accuracy, but caution should be maintained about the output being the "right" answer.

C. RECOMMENDATIONS FOR FOLLOW-ON WORK

Three areas of potential research would provide significant benefit to the output of this model, specifically, and to operators of the REMUS vehicle, in general.

1. Implementation of GPS Position Fixing in the Model

The vehicle is expected to have the capability to obtain a GPS fix in the near future. The impact of this technology has been hypothesized but not fully explored. It is thought that GPS fixes would significantly improve the accuracy of the vehicle, and intuition supports this claim. Implementation of this technology into a model such as the REMUS model in this thesis would go a long way toward answering questions about how much improvement can be expected. One consideration is explained here. Even with GPS the vehicle cannot get continuous position fixes. GPS signals do not currently penetrate water and the antenna proposed for the vehicle is not expected to extend above the surface unless the vehicle changes depth to make it happen. The upside is that when a fix

is recorded the accuracy of the fix would be much greater than the fixes obtained from triangulation. Perhaps the vehicle could be programmed to analyze its side-scan images real-time to determine when a potential object is located and then proceed to the surface to get a fix, or the vehicle might simply attempt to get a fix at set time intervals as is currently the case with triangulation. Discrete Event Simulation is ideally suited for exploring potential behaviors.

2. Vehicle Aspect Impact on Probability of Obtaining a Good Fix

A lot of data was analyzed to come up with distribution parameters for the range check random variable in the *RemusMover* component, see Appendix A for details. Time was a factor, however, and not enough data was evaluated to determine if vehicle aspect had a statistically significant impact on the probability of obtaining a good signal from the transducer. More analysis is needed in this area because the vehicle alternately turns toward and away from each transducer depending on where the next waypoint is or how the course is adjusted to regain track when a fix is obtained. Several mission playback files exist for data collection.

3. Developing a Graphical Interface for Operator Interaction

Many of the parameters in the model can be adjusted external to the execution class by utilizing JAVA property files. This method of interaction is fairly straightforward if the operator has a rudimentary knowledge of programming, but the average operator of the vehicle in the fleet probably does not possess the technical knowledge required to effectively change the parameters. A graphical interface would enhance the operator's ability to change parameters and observe the impact of the changes in a manner more easily interpreted than a large data output file processed by a commercial statistics package.

APPENDIX A. ANALYSIS OF PROBABILITY OF GOOD RETURN SIGNAL FROM TRANSDUCER

Arriving at an estimate for the distribution of the probability that a return signal from a transducer would be heard by the vehicle was not an easy process. Modeling sound behavior in water is a complicated science and accurate modeling requires much more information than is typically available to the operator or an analyst. The vehicle keeps track of sound speed information in its onboard computer and is continuously calculating the speed of sound, but these calculations are only accurate for the area the vehicle is currently transiting, not for any remote areas.

A significant amount of effort was expended in an attempt to quantify this error in order to increase the predictive accuracy of the resulting models. The first step involved gathering data and was the most time intensive step. The vehicle's mission playback files are available for past missions. A screenshot is provided in Figure 30 below. Highlighted areas include the vehicle information area in red, the track information section in blue and the transducer information section in green. Of interest in the data collection efforts were the transducer information area and the vehicle track area.

In order to collect meaningful data on transducer signal failure rates, the mission was played from start to finish and data were collected at every fix interval. This was very labor intensive and tedious. At each fix interval the playback was paused and the range information generated by the vehicle for each transducer was evaluated for accuracy. If a calculated range made sense then it was evaluated as good. An important distinction to be made is that the vehicle and the analyst did not always agree. There were instances where the vehicle assessed the signal as bad but the analyst assessed it as good based on other historical data, such as actual track reconstructed from known good fixes.

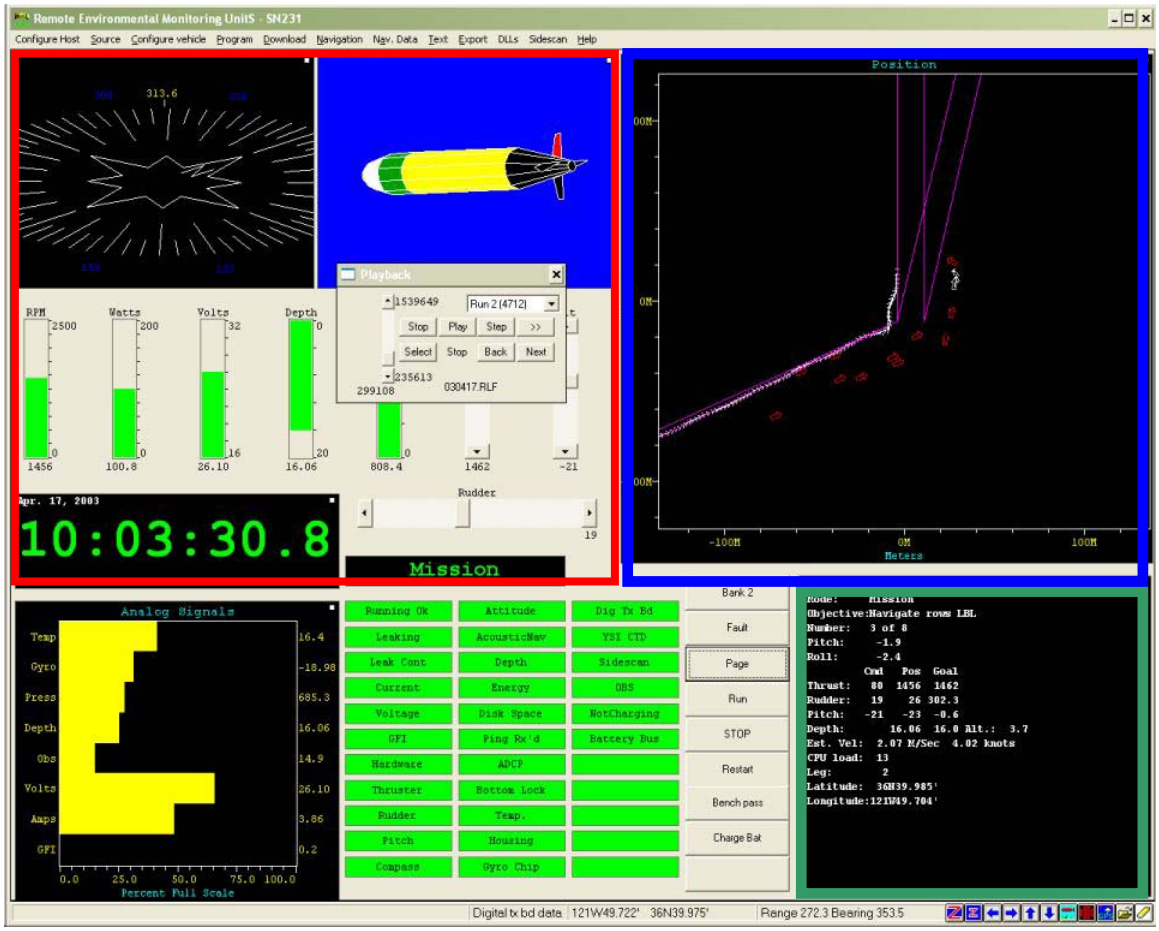


Figure 30. REMUS Mission Playback Screenshot

Figure 31 presents typical information available to the analyst following a good fix, as evaluated by the vehicle. Information includes calculated position in latitude and longitude, range from each transducer (DT1A and DT1B in this window) to the vehicle and other information which may be pertinent in other arenas. Each time the vehicle attempts to calculate a position fix a new window is generated and information must be evaluated.

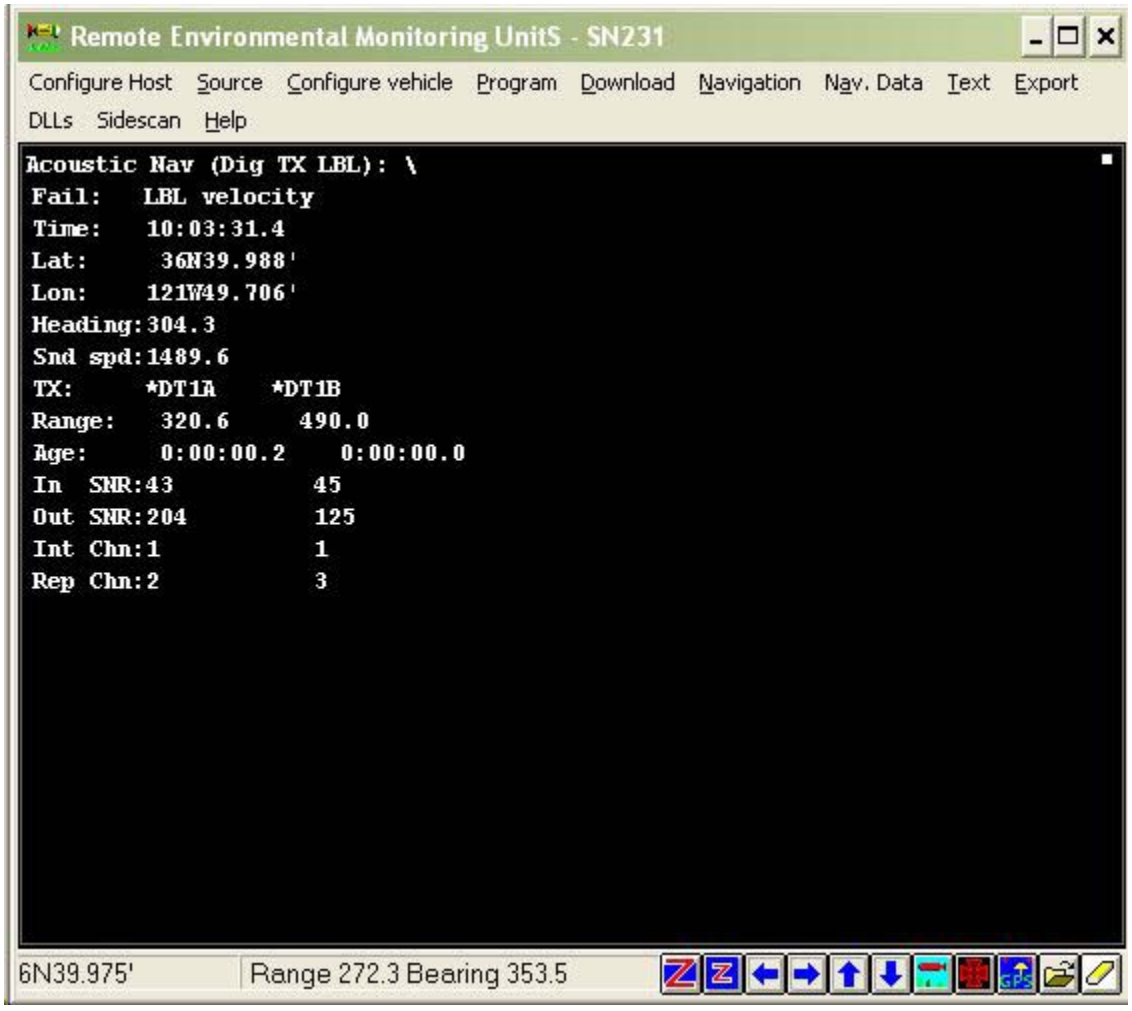


Figure 31. Transducer Information Available During Mission Playback – Good Fix

Figure 32 presents another transducer information screen this time showing a bad fix as assessed by the vehicle. In this instance the range computed to transducer DT1A is valid but the signal from transducer DT1B was never received, denoted by "timeout" in the fail category and the "(no data)" in the spot for transducer DT1B. In this case the information from transducer DT1A was valid and useful. The data need not be valid to be useful, however.

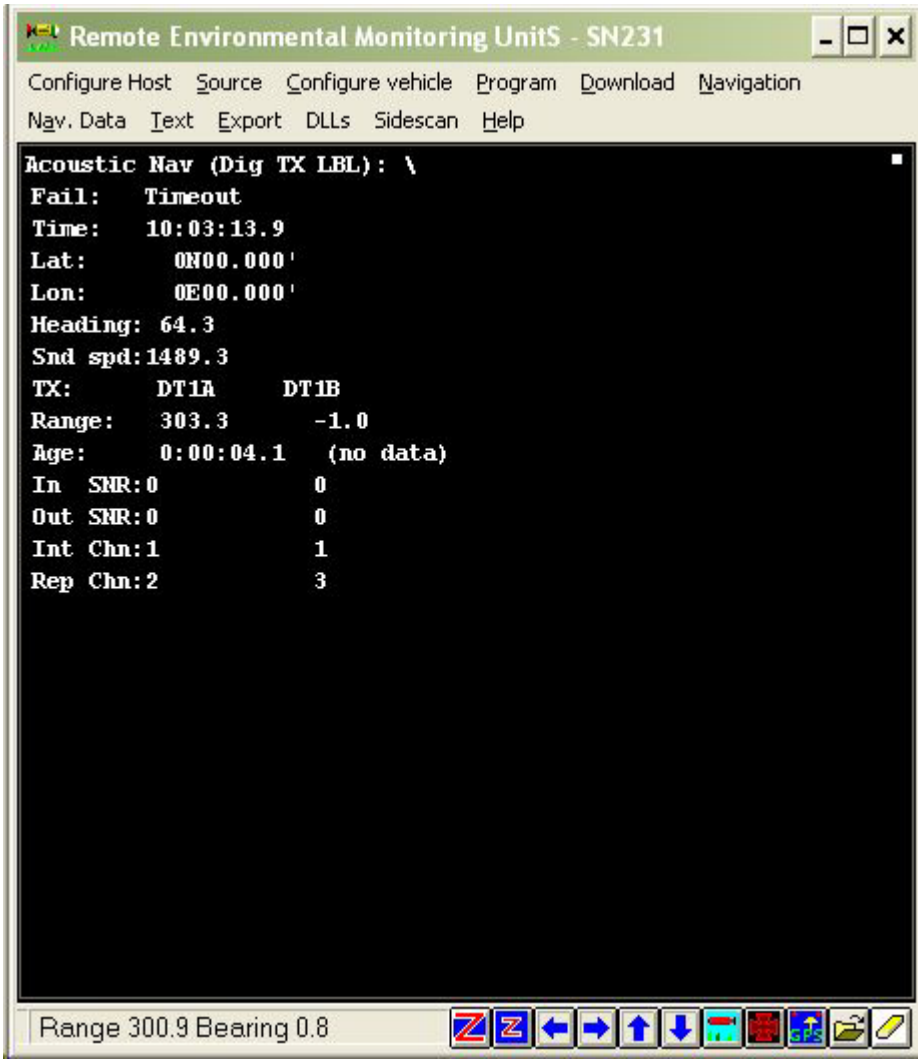


Figure 32. Transducer Information Screen Showing Bad Fix

Figure 33 presents the mission playback as a historical overlay of vehicle position and locations of fixes, both bad and good. A good fix is denoted by the white arrow and a bad fix is denoted by a red arrow. A red arrow denotes a bad fix even though two signals were returned. The vehicle assessed the fixes as bad. Note that the vehicle track, denoted by the white dashed line, shifts as soon as a good fix is obtained and the vehicle adjusts course to regain track. Each arrow represents one fix. There are some bad fixes which result in no arrow being added, such as when a signal is not returned.

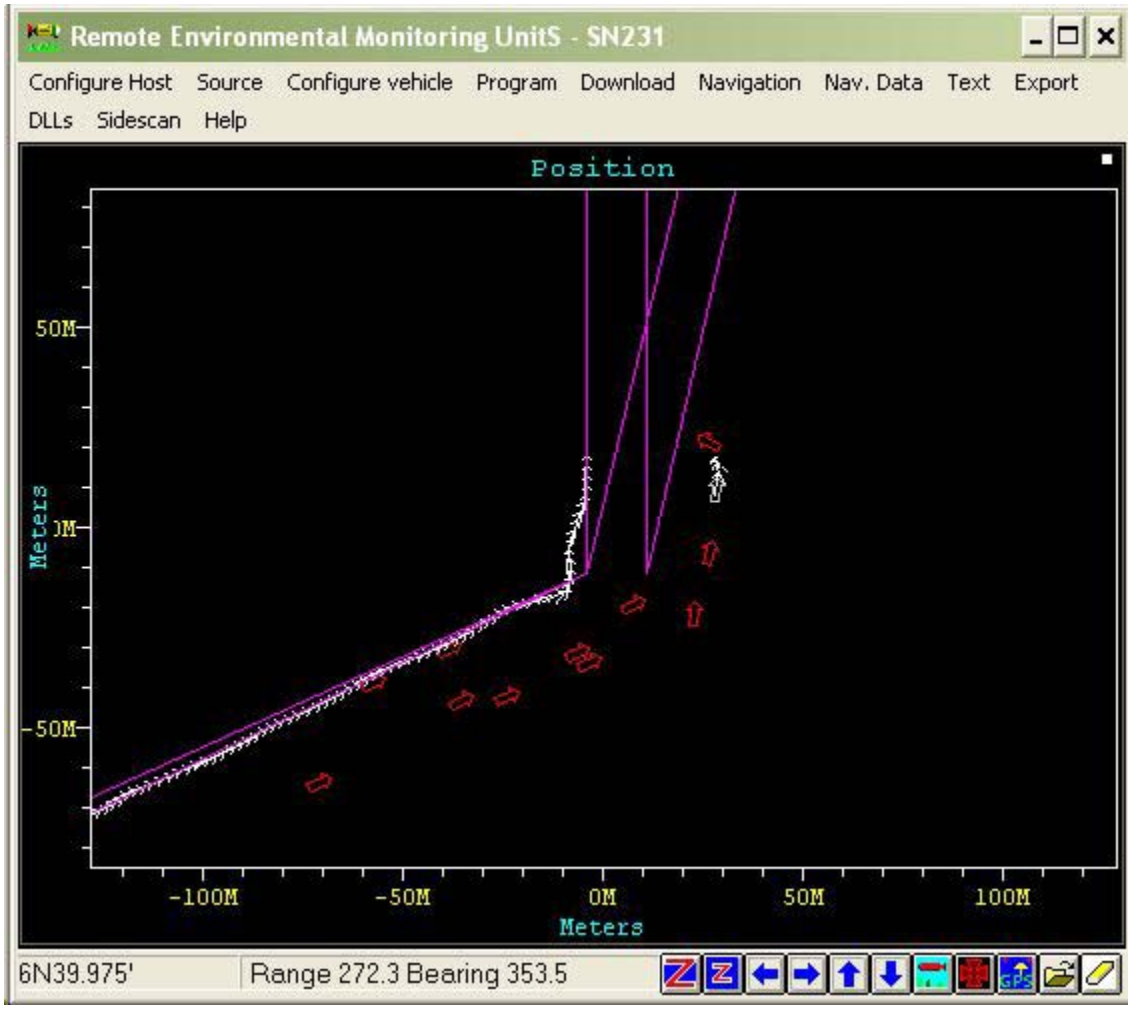


Figure 33. Mission Playback Window Showing Vehicle Track with Fixes

The range from the vehicle to either transducer was tracked by use of a coordinate system in an Excel spreadsheet. Fix information obtained from mission playback was compared to the expected values and assessed as being accurate or not. Comparing ranges at which the range calculations were accurate or not enabled an estimation of the distribution of the probability that a good signal was returned.

An attempt was also made to determine if the vehicle's relationship with the transducer had an impact on probability of a good return signal but no statistically significant relationship could be discerned.

After analysis of data from three missions totaling over three hours in length, the distribution of ranges was approximated by a normal distribution with mean 1000 feet

and standard deviation 500 feet. These number were picked to fit the distribution and also for simplicity in model implementation. The errors were not exactly normal but were close enough to allow simplification. Once the distribution was approximated another run of length 1.5 hours was used to check the parameters of the distribution with favorable results.

Mission playback files and the mission playback program are available from the Mechanical Engineering Department at the Naval Postgraduate School in Monterey, CA. A complete list of data collected in this effort is not included here due to the size of the file.

APPENDIX B. REMUS MOVER CODE

```
package remus;
import java.awt.geom.*;
import simkit.*;
import simkit.smdx.*;
import java.text.*;
import simkit.random.*;

/*
 * File: RemusMover.java
 * Created: August 21, 2003, 1:06 PM
 */

/**
 * <p>This is the main component in the REMUS simulation model. A
 * RemusMover maintains information about real and ideal locations
 * separately. The real location is the location of the mover as
 * affected by current, compass errors and other effects. The imaginary
 * location is where the vehicle thinks it is based on Dead Reckoning
 * techniques. If a RemusMover is using Transducers to fix position
 * then the vehicle utilizes triangulation to establish location. The
 * vehicle will attempt to regain the original track by steering for a
 * point on the track that is two fix intervals ahead of the current
 * location.</p>
 * @author Tim Allen
 * @version 1.0
 */
public class RemusMover extends SimEntityBase implements Mover {
  // class constants

  /**
   * Class constant corresponding to the origin of a standard x-y
   * coordinate system, namely (x,y) = (0.0,0.0).
   */
  protected static final Point2D ORIGIN = new Point2D.Double();
```

```
/**
 * Class constant used to format double values in output information.
 * The format is "0.000".
 */
protected static final DecimalFormat DF = new DecimalFormat("0.000");

// class variables
// instance variables

/**
 * This is the <CODE>RemusMover</CODE>'s maximum possible speed.
 * The value is in knots.
 */
protected double maxSpeed;

/**
 * The original location from which the <CODE>RemusMover</CODE>
 * started the run. The mover knows its original location with
 * certainty.
 */
protected Point2D originalLocation;

/**
 * The location the mover thinks it was at the last time it stopped.
 * Because of errors associated with navigation this position is not
 * necessarily the same as the lastRealStopLocation.
 */
protected Point2D lastIdealStopLocation;

/**
 * The location the mover was actually at the last tim it stopped.
 */
protected Point2D lastRealStopLocation;

/**
 * A representation of the current the vehicle is exposed to. The x
```

```
 * and y components give the portions of current in the x and y
 * direction.
 */
protected Point2D current;

/**
 * The ideal velocity is the velocity the vehicle thinks it is
 * making.
 */
protected Point2D idealVelocity;

/**
 * The real velocity is the velocity that the vehicle is actually
 * making. It takes into account the effects of current and compass
 * error.
 */
protected Point2D realVelocity;

/**
 * The destination the vehicle thinks it is moving towards.
 */
protected Point2D idealDestination;

/**
 * The destination the vehicle is actually headed towards.
 */
protected Point2D realDestination;

/**
 * The ultimate destination is the place the vehicle is supposed to
 * go on this leg.
 */
protected Point2D ultimateDestination;

/**
 * The position the vehicle started this leg from. Used to calculate
 * track intercepts in the doFix method.
```

```
 */
protected Point2D anchorLocation;

/**
 * The sim time that the current move started.
 */
protected double startMoveTime;

/**
 * Move time is the amount of time necessary to move to the
 * destination at the given velocity.
 */
protected double moveTime;

/**
 * Time in seconds between vehicle interogation of the transducers.
 */
protected double pingInterval;

/**
 * True if the vehicle is using transducers, false otherwise.
 */
protected boolean usingTransducers;

/**
 * Current movement status of the vehicle.
 */
protected MovementState movementState;

/**
 * A double value which is interpreted as a percentage of the
 * heading.
 */
protected double compassError;

/**
 * Used to fix vehicle position.
```

```
 */
protected Transducer transducerA;

/**
 * Used to fix vehicle position.
 */
protected Transducer transducerB;

protected final Object[] param;

/**
 * Used to generate a comparison value when determining if a signal
 * from the transducer is heard by the vehicle.
 */
protected RandomVariate rvRangeCheck;

// class methods

/*
 * This private helper method will adjust the given "vector" in a
 * Point2D object by rotating the coordinates by a percentage of
 * 2Pi, given by the double error. Divide by zero problems are
 * handled by the Math class. Rotation follows math conventions,
 * that is, a negative error will lower theta thus giving a heading
 * error to the right.
 */
private static Point2D calculateCompassError(Point2D velocity,
                    double error) {
  double xComp = velocity.getX();
  double yComp = velocity.getY();
  double hyp = Math.sqrt(xComp*xComp + yComp*yComp);
  double theta = Math.atan2(yComp, xComp);
  theta += error * 2 * Math.PI;
  xComp = hyp * Math.cos(theta);
  yComp = hyp * Math.sin(theta);

  return new Point2D.Double(xComp, yComp);
```

```
}

/**
 * Helper method to format Point2D objects.
 * @param point Point to be formatted.
 * @param form Rule for format.
 * @return Formatted Point.
 */
public static String formatPoint(Point2D point, DecimalFormat form) {
   StringBuffer buf = new StringBuffer('[');
   buf.append(form.format(point.getX()));
   buf.append(',');
   buf.append(' ');
   buf.append(form.format(point.getY()));
   buf.append(']');

   return buf.toString();
}

/**
 * Helper method to format Point2D objects. Uses class default
 * format object.
 * @param point Point to be formatted.
 * @return formatted point.
 */
public static String formatPoint(Point2D point) {
   return formatPoint(point, DF);
}

// constructor methods

/**
 * Creates new instance of RemusMover with an initial location and
 * max speed. Sets compassError to 0.0 by default. Sets current to
 * (0.0, 0.0) by default. Movement state is set to stopped. Poisiton
 * fixing is off by default.
 * @param location The new instances initial location.
```

```
 * @param maxSpeed The new instances maximum speed.
 */
public RemusMover(Point2D location, double maxSpeed) {
  setCompassError(0.0);
  setName("Default");
  originalLocation = (Point2D) location.clone();
  anchorLocation = (Point2D) location.clone();
  this.maxSpeed = maxSpeed;
  lastIdealStopLocation = (Point2D) originalLocation.clone();
  lastRealStopLocation = (Point2D) originalLocation.clone();
  idealVelocity = (Point2D) ORIGIN.clone();
  realVelocity = (Point2D) ORIGIN.clone();
  setCurrent((Point2D)ORIGIN.clone());
  setMovementState(MovementState.STOPPED);
  transducerA = new Transducer("defaultA", 0.0, 0.0);
  transducerB = new Transducer("defaultB", 0.0, 0.0);
  setUsingTransducers(false);
  param = new Object[] {this};
  rvRangeCheck = null;
}

/**
 * Creates new instance of RemusMover and sets name.
 * @param name
 * @param location
 * @param maxSpeed
 */
public RemusMover(String name, Point2D location, double maxSpeed) {
  this(location, maxSpeed);
  setName(name);
}

/**
 * Creates new instance of RemusMover and takes argument for
 * transducers.
 * @param name
 * @param location
```

```
     * @param maxSpeed
     * @param a Transducer A
     * @param b Transducer B
     * @param ping The ping interval in seconds
     * @param rv The <code>RandomVariate</code> object used for range check logic
     * in the Fix method.
     */
    public RemusMover(String name, Point2D location, double maxSpeed,
      Transducer a, Transducer b, double ping, RandomVariate rv) {
        this(name, location, maxSpeed);
        transducerA = a;
        transducerB = b;
        setPingInterval(ping);
        setUsingTransducers(true);
        setRVRangeCheck(rv);
  }

    // instance methods

    /**
     * Does nothing, yet.
     */
    public void accelerate(Point2D acceleration) {
    }

    /**
     * Does nothing, yet.
     */
    public void accelerate(Point2D acceleration, double speed) {
    }

    /**
     * When this event is executed the mover stops at current location
     * and the movement state is set to paused.
     * @param mover The Mover for which the event is scheduled.
     */
    public void doEndMove(Moveable mover) {
```

```
  if (mover == this) {
    anchorLocation = ultimateDestination;
    stopHere();
    setMovementState(MovementState.STOPPED);
    interrupt("Fix");
    interrupt("Intercept");
  }
}

/**
 * When this event is executed the endMove event is scheduled as
 * long as the idealLocation is not null. Movement state is set to
 * cruising.
 * @param mover The Mover that the event is scheduled for.
 */
public void doStartMove(Moveable mover) {
  if (mover == this) {
    if (idealDestination != null) {
      waitDelay("EndMove", moveTime, param, 1.0);
      /* Don't want to schedule a fix event unless the vehicle
       * is using transducers and it is paused. If the fix event
       * isn't scheduled here then it will never be scheduled
       * again.*/
      if(isUsingTransducers() && movementState != MovementState.PAUSED) {
        waitDelay("Fix", getPingInterval(), param);
        ultimateDestination = idealDestination;
      }
    }
    setMovementState(MovementState.CRUISING);
  }
}

/**
 * Not yet used.
 * @return A new Point2D.Double set to (0.0, 0.0).
 */
public Point2D getAcceleration() {
```

```
  return new Point2D.Double();
}

/**
 * Returns the actual location of RemusMover as affected by current
 * and compass errors.
 * @return A clone of the actual location of the RemusMover for the
 * current SimTime.
 */
public Point2D getLocation() {
  if (isMoving()) {
    return new Point2D.Double(lastRealStopLocation.getX()
      + (Schedule.getSimTime() - startMoveTime)
      * getVelocity().getX(), lastRealStopLocation.getY()
      + (Schedule.getSimTime() - startMoveTime)
      * getVelocity().getY());
  }
  else {
    return (Point2D) lastRealStopLocation.clone();
  }
}

/**
 * Returns the ideal location of RemusMover unaffected by current
 * and compass errors.
 * @return A clone of the ideal location of the RemusMover for the
 * current SimTime.
 */
public Point2D getIdealLocation() {
  if (isMoving()) {
    return new Point2D.Double(lastIdealStopLocation.getX()
      + (Schedule.getSimTime() - startMoveTime)
      * getIdealVelocity().getX(), lastIdealStopLocation.getY()
      + (Schedule.getSimTime() - startMoveTime)
      * getIdealVelocity().getY());
  }
  else {
```

```
    return (Point2D) lastIdealStopLocation.clone();
  }
}

/**
 * Sets the current to the value of newCurrent. The current is
 * represented by a Point2D object. The X value corresponds to the
 * component of current in the X direction in a two dimensional grid.
 * The case is similar for Y.
 * @param newCurrent
 */
public void setCurrent(Point2D newCurrent) {
  current = (Point2D) newCurrent.clone();
}

/**
 * Returns the current for this instance. The current is represented
 * by a Point2D object. The X value corresponds to the component of
 * current in the X direction in a two dimensional grid. The case is
 * similar for Y.
 * @return current
 */
public Point2D getCurrent() {
  return (Point2D) current.clone();
}

/**
 * Returns maximum possible speed.
 * @return maxSpeed
 */
public double getMaxSpeed() {
  return maxSpeed;
}

/**
 * Returns movementState. See the MovementState class for
 * description of the constants.
```

```
 * @return movementState
 */
public MovementState getMovementState() {
  return movementState;
}

/**
 * Returns the realVelocity of the RemusMover, including effects of
 * current and gyro error.
 * @return realVelocity
 */
public Point2D getVelocity() {
  return (Point2D) realVelocity.clone();
}

/**
 * Returns the idealVelocity of the RemusMover. This is the velocity
 * of the vehicle with no errors or effects of current.
 * @return idealVelocity
 */
public Point2D getIdealVelocity() {
  return (Point2D) idealVelocity.clone();
}

/**
 * Used to set the boolean usingTransducers, true if transducers are
 * used, false otherwise.
 * @param f True if transducers used, false otherwise.
 */
public void setUsingTransducers(boolean f) {
  usingTransducers = f;
}

/**
 * Returns true if fixing position with transducers, false otherwise.
 * @return usingTransducers
 */
```

```
public boolean isUsingTransducers() {
  return usingTransducers;
}

/**
 * Sets ping interval for transducers.
 * @param ping A double value corresponding to ping interval, in
 * seconds.
 */
public void setPingInterval(double ping) {
  pingInterval = ping;
}

/**
 * Returns ping interval in seconds.
 * @return pingInterval
 */
public double getPingInterval() {
  return pingInterval;
}

/**
 * Sets the Random Variate object used to perform raneg checks in
 * the Fix method.
 * @param rv
 */
public void setRVRangeCheck(RandomVariate rv) {
  this.rvRangeCheck = rv;
}

/**
 * Access the Random Variate object used to perform range check
 * functions.
 * @return The RandomVector object used for checking if transducers
 * are range for position fix.
 */
public RandomVariate getRVRangeCheck() {
```

```
  return rvRangeCheck;
}

/**
 * Returns the status of moving.
 * @return isMoving True if moving, false otherwise.
 */
public boolean isMoving() {
  return Math.abs(idealVelocity.getX()) > 0.0
     || Math.abs(idealVelocity.getY()) > 0.0;
}

/**
 * Not yet implemented and no plans to.
 * @param location
 */
public void magicMove(Point2D location) throws MagicMoveException {
}

/**
 * Checks if requested velocity exceeds maxSpeed. Adjusts ideal
 * velocity for current and assignes value as real velocity.
 * Schedules the startMove event and sets movement state to STARTING.
 * @param desiredVelocity
 */
public void move(Point2D desiredVelocity) {
  double desiredSpeed = desiredVelocity.distance(ORIGIN);
  if (desiredSpeed <= 0.0) {
    return;
  }
  if ( desiredSpeed > maxSpeed) {
    desiredVelocity = Math2D.scalarMultiply(maxSpeed / desiredSpeed,
      desiredVelocity);
  }
  realDestination = null;
  idealDestination = null;
  lastRealStopLocation = getLocation();
```

```
    lastIdealStopLocation = getIdealLocation();
    startMoveTime = Schedule.getSimTime();
    idealVelocity = desiredVelocity;
    realVelocity = Math2D.add(calculateCompassError(desiredVelocity,
      compassError), current);
    waitDelay("StartMove", 0.0, param);
    setMovementState(MovementState.STARTING);
  }

  /**
   * Resets variables to original values. Specifically, resets
   * locations to clones of original locations, velocities to zero,
   * destinations to null and compassError to 0.0.
   */
  public void reset() {
    super.reset();
    setMovementState(MovementState.STOPPED);
    lastIdealStopLocation = (Point2D) originalLocation.clone();
    lastRealStopLocation = (Point2D) originalLocation.clone();
    realVelocity  = new Point2D.Double();
    idealVelocity  = new Point2D.Double();
    startMoveTime = Schedule.getSimTime();
    idealDestination = null;
    realDestination = null;
    compassError = 0.0;
  }

  /**
   * Additional moveTo that accepts destination. MaxSpeed is assumed.
   * @param destination
   */
  public void moveTo(Point2D destination) {
    moveTo(destination, maxSpeed);
  }

  /**
   * When executed checks to ensure a destination exists and
```

```
 * cruisingSpeed is greater than zero. Calculates new ideal and
 * real destinations. Corrects real velocity for compass error and
 * current effects. Schedules startMove event.
 * @param destination Where the Mover is going
 * @param cruisingSpeed Speed of motion, unless cruisingSpeed is
 * greater than maxSpeed, in which case maxSpeed is used.
 */
public void moveTo(Point2D destination, double cruisingSpeed) {
  if (destination == null || cruisingSpeed <= 0.0) { return; }
  if (isMoving()){
    pause();
  }
  cruisingSpeed = Math.min(cruisingSpeed, maxSpeed);
  this.idealDestination = destination;
  double distance = destination.distance(this.getIdealLocation());
  moveTime = distance / cruisingSpeed;
  idealVelocity.setLocation((idealDestination.getX()
    - lastIdealStopLocation.getX()) / moveTime, (idealDestination.getY()
    - lastIdealStopLocation.getY())/moveTime);
  realVelocity = Math2D.add(calculateCompassError(idealVelocity,
    compassError), current);
  double realX = lastRealStopLocation.getX() + realVelocity.getX() * moveTime;
  double realY = lastRealStopLocation.getY() + realVelocity.getY() * moveTime;
  realDestination = new Point2D.Double(realX, realY);
  startMoveTime = Schedule.getSimTime();
  waitDelay("StartMove", 0.0, new Object[] { this });
}

/**
 * Formatting method. Returns object's name, original location and
 * max speed
 * @return A String containing output information.
 */
public String paramString() {
  return this.getName() + " " + originalLocation + " " + maxSpeed;
}
```

```
/**
 * Stops the Mover's motion, sets movementState to PAUSED.
 */
public void pause() {
  stopHere();
  setMovementState(MovementState.PAUSED);
}

/**
 * Stops the Mover's motion, sets movementState to STOPPED.
 */
public void stop() {
  stopHere();
  setMovementState(MovementState.STOPPED);
}

/**
 * Sets movementState to state, executes firePropertyChange for
 * movementState.
 * @param state New movementState.
 */
protected void setMovementState(MovementState state) {
  MovementState oldState = getMovementState();
  movementState = state;
  firePropertyChange("movementState", oldState, movementState);
}

/**
 * Sets compassError. Fires property change for compass error.
 * @param error The compassError as a percentage of heading.
 */
public void setCompassError(double error) {
  firePropertyChange("compassError", new Double(compassError),
    new Double(error));
  compassError = error;
}
```

```
/**
 * Stops Mover's motion. Interupts the endMove event for this mover.
 * Sets ideal and real locations. Resets velocities to zero.
 */
protected void stopHere() {
  lastIdealStopLocation = getIdealLocation();
  lastRealStopLocation = getLocation();
  if (idealVelocity == null) {
    idealVelocity = new Point2D.Double();
    realVelocity = new Point2D.Double();
  }
  else {
    realVelocity.setLocation(ORIGIN);
    idealVelocity.setLocation(ORIGIN);
  }
  startMoveTime = Schedule.getSimTime();
  interrupt("EndMove", param);
}

/**
 * Formatting method to output object information as a String.
 * @return Real and ideal locations of Mover.
 */
public String toString() {
  Point2D idealLoc = getIdealLocation();
  Point2D realLoc = getLocation();
  return this.getName() + " I (" + DF.format(idealLoc.getX())
    + "," + DF.format(idealLoc.getY()) +") ["
    + DF.format(this.getIdealVelocity().getX()) + ","
    + DF.format(this.getIdealVelocity().getY()) + "], R " + "("
    + DF.format(realLoc.getX()) + "," + DF.format(realLoc.getY())
    + ") [" + DF.format(this.getVelocity().getX()) + ","
    + DF.format(this.getVelocity().getY()) + "]";
}

/**
 * Returns the difference between ideal and real locations for the
```

```
 * Mover at current time.
 * @return The distance between real and ideal locations.
 */
public double getDistanceDifference() {
  return getLocation().distance(getIdealLocation());
}

/**
 * Generates a comparison value and determines if return signal is
 * heard by vehicle. If the vehicle i smoving a new position is
 * calculated by triangulation and the position is updated.
 * Previously scheduled intercept events are interrupted. New
 * movement orders are issued to regain track and new intercept and
 * fix events are scheduled. If no return signal is heard then only
 * the next fix event is scheduled. If the vehicle is not moving
 * then nothing happens.
 *
 * @param mover The Mover object the event is scheduled for.
 */
public void doFix(Moveable mover) {
  double compareValue = rvRangeCheck.generate();
  double distA = getLocation().distance(transducerB.getLocation());
  double distB = getLocation().distance(transducerA.getLocation());
  if (distA < compareValue && distB < compareValue) {
    double x = 0.0;
    double y = 0.0;
    double angleA = 0.0;
    double distC = transducerA.getIdealLocation().distance(
      transducerB.getIdealLocation());
    Point2D vecC = Math2D.subtract(transducerB.getLocation(),
      transducerA.getLocation());
    Point2D unitVecC = Math2D.scalarMultiply(1.0 / Math2D.norm(vecC), vecC);
    Point2D vecB = Math2D.subtract(getLocation(), transducerA.getLocation());
    Point2D unitVecB = Math2D.scalarMultiply(1.0 / Math2D.norm(vecB), vecB);
    double check = (distB * distB + distC * distC - distA * distA)
      / (2.0 * distB * distC);
    if (check >= 1.0) {
```

```
    angleA = 0.0;
  } else if (check <= -1.0) {
    angleA = 180.0;
  } else if (vecB.getY() > vecC.getY()) {
    angleA = Math.acos(check);
  } else {
    angleA = -Math.acos(check);
  }
  double angleAprime = Math.atan2(vecC.getY(), vecC.getX());
  double angleTot = angleA + angleAprime;
  x = transducerA.getLocation().getX() + distB * Math.cos(angleTot);
  y = transducerA.getLocation().getY() + distB * Math.sin(angleTot);
  Point2D vecA = Math2D.subtract(ultimateDestination, anchorLocation);
  vecB = Math2D.subtract(new Point2D.Double(x, y), anchorLocation);
  Point2D projAb = Math2D.add(anchorLocation,
    Math2D.scalarMultiply(Math2D.innerProduct(vecA, vecB)
    / Math2D.normSq(vecA), vecA));
  Point2D trackVel = Math2D.scalarMultiply(1.0 / Math2D.norm(vecA)
    * maxSpeed, vecA);
  Point2D desiredLocation = Math2D.add(projAb,
  Math2D.scalarMultiply(2.0 * getPingInterval(), trackVel));
  if (movementState == MovementState.CRUISING) {
    interrupt("Intercept");
    this.pause();
    lastIdealStopLocation.setLocation(x, y);
    if (projAb.distance(desiredLocation)
      < projAb.distance(ultimateDestination)) {
        moveTo(desiredLocation);
    }
    else {
      moveTo(ultimateDestination);
    }
    waitDelay("Fix", getPingInterval(), param);
    waitDelay("Intercept", 2.0 * getPingInterval());
  }
}
else {
```

```
        waitDelay("Fix", getPingInterval(), param);
      }
    }

    /**
     * Method performs actions for when the vehicle regains track again.
     * Movement is paused and the vehicle is ordered to its ultimate
     * destination. The Fix event is interrupted and the next Fix event
     * is scheduled after a delay equal to the ping interval.
     */
    public void doIntercept() {
      this.pause();
      this.moveTo(ultimateDestination);
      interrupt("Fix");
      waitDelay("Fix", getPingInterval(), param);
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. TRANSDUCER CODE

```
package remus;
import simkit.*;
import simkit.smdx.*;
import simkit.random.*;
import java.awt.geom.*;
import java.text.DecimalFormat;

/*
 * File: Transducer.java
 * Created: August 7, 2003, 4:19 PM
 */

/**
 * <p><code>Transducer</code> objects contain and provide position information.
 * They are used by the REMUS vehicle to obtain a fix of position.
 * <code>Transducer</code> objects are positioned in a coordinate field and then
 * provide their position to other objects incorporating random error.</p>
 * @author Tim Allen
 */
public class Transducer extends SimEntityBase {
  // class constants
  // class variables

  private static DecimalFormat f = new DecimalFormat("0.00");

  // instance variables

  /**
   * The <code>Transducer</code>'s real location. This is the <code>
   * Transducer</code>'s ideal location offset by some error to reflect
   * uncertainty in dropping placing the <code>Transducer</code>.
   */
  protected Point2D realLocation;
```

```
/**
 * The <code>Transducer</code>'s ideal location. This is the location the
 * vehicle thinks the <code>Transducer</code> is at.
 */
protected Point2D idealLocation;

/**
 * A <code>RandomVector</code> object used to alter the <code>Transducer
 * </code>'s real position when queried to reflect errors introduced by
 * <code>Transducer</code> sway, current, etc.
 */
protected RandomVector rv;

// class methods
// constructor methods

/**
 * Zero-parameter constructor. <code>Transducer</code> is placed at (0,0) and
 * given name "default". The position location error is not used
 * in this constructor. The <code>RandomVector</code> instance is initailized
 * to null.
 */
public Transducer() {
  this("default", 0.0, 0.0, null, null);
}

/**
 * An instance of <code>Transducer</code> is created with name and location
 * provided. The position uncertainty <code>RandomVector</code> is
 * initialized to null.
 * @param name
 * @param point A <code>Point2D</code> object used as the <code>Transducer
 * </code>'s location.
 */
public Transducer(String name, Point2D point) {
  this(name, point.getX(), point.getY(), null, null);
}
```

```
/**
 * An instance of <code>Transducer</code> is created with name and location
 * provided. The position uncertainty <code>RandomVector</code> is
 * initialized to null.
 * @param name
 * @param x X location.
 * @param y Y location.
 */
public Transducer(String name, double x, double y) {
  this(name, x, y, null, null);
}

/**
 * An instance of <code>Transducer</code> is created with name and location
 * provided. The <code>Transducer</code>'s initial location is offset by the
 * gps parameter. The <code>RandomVector</code> object is used to produce
 * position uncertainty.
 * @param name
 * @param x X location.
 * @param y Y location.
 * @param gps A double array with two elements, one for x and one
 * for y, used to offset the initial location for simulating drop
 * errors.
 * @param rv A <code>RandomVector</code> object used to produce position
 * uncertainty when the <code>Transducer</code> is queried for its location.
 */
public Transducer(String name, double x, double y, double[] gps, RandomVector rv) {
  idealLocation = new Point2D.Double(x, y);
  setName(name);
  if(gps != null) {
    realLocation = new Point2D.Double(x + gps[0], y + gps[1]);
  }
  else {
    realLocation = new Point2D.Double(x, y);
  }
  if(rv != null) {
```

```
    this.rv = rv;
  }
}

/**
 * An instance of <code>Transducer</code> is created with name and location
 * provided. The <code>Transducer</code>'s initial location is offset by the
 * gps parameter. The <code>RandomVector</code> object is used to produce
 * position uncertainty.
 * @param name
 * @param loc A Point2D object representing the location.
 * @param gps A double array with two elements, one for x and one
 * for y, used to offset the initial location for simulating drop
 * errors.
 * @param rv A <code>RandomVector</code> object used to produce position
 * uncertainty when the <code>Transducer</code> is queried for its location.
 */
public Transducer(String name, Point2D loc, double[] gps, RandomVector rv) {
  this(name, loc.getX(), loc.getY(), gps, rv);
}

// instance methods

/**
 * Returns the <code>Transducer</code> ideal location, or where the vehicle
 * thinks th etransducer is.
 * @return <code>Point2D</code> object representing the ideal location.
 */
public Point2D getIdealLocation() {
  return idealLocation;
}

/**
 * Returns the <code>Transducer</code> real position as affected by current
 * and wave action and drop error.
 * @return <code>Point2D</code> object representing <code>Transducer</code>
 * real location.
```

```
   */
  public Point2D getLocation() {
    if(rv != null) {
      double[] xy = rv.generate();
      Point2D offset = new Point2D.Double(xy[0], xy[1]);
      return Math2D.add(realLocation, offset);
    }
    return (Point2D)realLocation.clone();
  }

  /**
   * Set the <code>Transducer</code>'s real location.
   * @param p p[0] is the x value, p[1] is the y value
   */
  public void setLocation(double[] p) {
    realLocation = Math2D.add(new Point2D.Double(p[0], p[1]), idealLocation);
  }

  /**
   * Set the <code>RandomVector</code> object used to generate real locations.
   * @param rv <code>RandomVector</code> object used for position generation
   */
  public void setRV(RandomVector rv) {
    this.rv = rv;
  }

  /**
   * Formatting of <code>Transducer</code> - name, location info.
   * @return String representation of object info.
   */
  public String toString() {
    return "Transducer " + getName() + " at location ("
    + idealLocation.getX() + ", " + idealLocation.getY() + ")";
  }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. RANGE FINDER CODE

```
package remus;
import simkit.*;
import simkit.smdx.*;
import java.awt.geom.*;
import java.text.*;

/*
 * File: RangeFinder.java
 * Created: November 17, 2003, 10:45 AM
 */

/**
 * <p>A <code>RangeFinder</code> is used to monitor for the detection event
 * and when that event is heard then mine location information is output in a
 * format suitable for data analysis.</p>
 *
 * @author Tim Allen
 */
public class RangeFinder extends SimEntityBase{
  // class constants
  // class variables
  // instance variables

  /**
   * Keeps a copy of the </code>RemusMover</code> in order to calculate the
   * offset information.
   */
  protected RemusMover searcher;

  /**
   * Keeps a copy of the <code>Mover</code> (the mine) in order to claculate
   * offset information.
   */
  protected Mover target;
```

```
/**
 * Keeps track of whether detection has occured since the last reset
 * invocation.
 */
protected boolean detected;

/*
 * Used internally for formatting output.
 */
private DecimalFormat f = new DecimalFormat("0.00");

// class methods
// constructor methods

/**
 * Creates instance of <code> RangeFinder</code> associated with searcher and
 * target. Initially, detected set to false.
 * @param searcher The <code>RemusMover</code> associated with this <code>
 * RangeFinder</code>
 * @param target The <code>Mover</code> associated with this <code>
 * RangeFinder</code>
 */
public RangeFinder(RemusMover searcher, Mover target) {
  this.searcher = searcher;
  this.target = target;
  detected = false;
}
// instance methods

/**
 * Resets detected to false.
 */
public void reset() {
  detected = false;
}
```

```
  /**
   * If this is the first time an object is detected then an offset is
   * calculated and output to the System.out stream.
   * @param contact
   */
  public void doDetection(Moveable contact) {
    if(!detected) {
      detected = true;
      Point2D searcherLoc = searcher.getLocation();
      Point2D targetLoc = target.getLocation();
      Point2D offset = new Point2D.Double(targetLoc.getX() - searcherLoc.getX(),
        targetLoc.getY() - searcherLoc.getY());
      System.out.print(f.format(searcher.getIdealLocation().getX() + offset.getX())
        + " " + f.format(searcher.getIdealLocation().getY() + offset.getY()));
    }
  }

  /**
   * Returns true if detection has occurred, false otherwise.
   * @return detected
   */
  public boolean getDetected() {
    return detected;
  }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. REMUS SENSOR MEDIATOR CODE

```
package remus;
import simkit.*;
import simkit.smdx.*;
import java.util.*;
import java.awt.geom.*;

/*
 * File: RemusSensorMediator.java Created: February 22, 2004, 6:36 PM
 */

/**
 * <p>Mediator used by the <code>RemusMover</code> component to model detection
 * when obstacle is at Closest Point of Approach (CPA) with vehicle. The CPA
 * method most accurately approximates the method used by the human operator
 * when looking at mission playback logs.</p>
 *
 * @author Tim Allen
 */
public class RemusSensorMediator extends SimEntityBase implements SensorTargetMediator {

  // class constants
  // class variables
  // instance variables

  /**
   * Used to keep track of contacts once they enter range.
   */
  protected Map contacts;

  // class methods
  // constructor methods

  /**
   * Creates new RemusSensorMediator.
```

```
 */
public RemusSensorMediator() {
  contacts = new WeakHashMap();
}

// instance methods

/**
 * When this event is heard, schedule the detection for the sensor based on
 * the detection algorithm - by calculating time of CPA. The sensor is passed
 * an instance of a Contact, which is a doppleganger for the actual target.
 * Note that the waitDelay() is invoked directly on the sensor. This makes
 * the sensor the "source" of the SimEvent, and allows listeners to the
 * sensor to be able to hear it.
 *
 * @param sensor The sensor whose range was entered
 * @param target The target that entered the sensor's range
 */
public void doEnterRange(Sensor sensor, Mover target) {
  if (this == SensorTargetMediatorFactory.getInstance().getMediatorFor(
    sensor.getClass(), target.getClass())) {
      Object contact = contacts.get(target);
      if (contact == null) {
        contact = new Contact(target);
        contacts.put(target, contact);
      }
      double speed = Math2D.norm(sensor.getVelocity());
      double distance = sensor.getLocation().distance(target.getLocation());
      Point2D relativeDir = Math2D.subtract(target.getLocation(),
        sensor.getLocation());
      Point2D realVel = sensor.getVelocity();
      double theta = Math.acos(Math2D.innerProduct(relativeDir, realVel)
        / (Math2D.norm(relativeDir) * Math2D.norm(realVel)));
      double rangeToDetect = distance * Math.cos(theta);
      double timeToDetect = rangeToDetect / speed;
      sensor.waitDelay("Detection", timeToDetect, new Object[] { contact});
  }
```

```
  }

  /**
   * When the range is exited, what happens in general depends on the
   * undetection rule. In this case, simply schedule an undetection. Like the
   * Detection event, the Undetection event is scheduled directly on the
   * sensor, for the same reason.
   *
   * @param sensor The sensor whose range was exited
   * @param target The target that exited the sensor's range
   */
  public void doExitRange(Sensor sensor, Mover target) {
    if (this == SensorTargetMediatorFactory.getMediator(sensor.getClass(),
        target.getClass())) {
          Object contact = contacts.get(target);
          sensor.waitDelay("Undetection", 0.0, new Object[] { contact});
    }
  }

  /**
   * Do nothing
   *
   * @param evt The heard PropertyChangeEvent
   */
  public void propertyChange(java.beans.PropertyChangeEvent evt) {
  }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Box, E., W. Hunter, J. Hunter (1978). Statistics for Experimenters. New York: John Wiley & Sons.

Buss, A. (2001). Basic Event Graph Modeling. *Simulation News Europe.* 31:1-6.

Buss, A. (2002). Component Based Simulation Modeling with Simkit. In *Proceedings of the 2002 Winter Simulation Conference*, ed E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Buss, A. (2003). Simple Movement and Detection in Discrete-Event Simulation. *OA3202 Class Notes.* Naval Postgraduate School, Monterey, CA.

Bratley, P., Fox, B., Schrage, L. (1983). A Guide to Simulation. New York: Springer-Verlag.

Childs, M. (March 2002). An Exploratory Analysis of Waterfront Force Protection Measures Using Simulation. M.S. Thesis, Naval Postgraduate School, Monterey, CA.

Denne, W. (1951). Magnetic Compass Deviation and Correction. Glasgow: Brown, Son and Ferguson Limited.

Eisman, D. (2003, March 22). Navy ships seize boats carrying mines in Iraqi port. The Virginian-Pilot. Retrieved November 19, 2003, from http://www.globalsecurity.org/org/news/2003/030322-mineboats01.htm.

Hamilton, L. (1992). Regression with Graphics, A second Course in Applied Statistics. Belmont: Duxbury Press.

Kleijnen, J. P. C., Sanchez, S. M., Lucas, T. W., Cioppa, (2004). A User's Guide to the Brave New World of Simulation Experiments. Working paper, Department of Information Management/Center for Economic Research (CentER), Tilburg University, Tilburg, The Netherlands.

Law, A, Kelton, W. (2000). Simulation Modeling and Analysis (3rd Edition). Boston: Mcgraw Hill.

Nawara, T. (September 2003). Exploratory Analysis of Submarine Tactics for Mine Detection and Avoidance. M.S. Thesis, Naval Postgraduate School, Monterey, CA.

Ocean Technology Systems. Sea State Chart. Retrieved April 2004, from http://www.oceantechnologysystems.com/sea_conditions.html.

Schruben, L. (1983). Simulation Modeling with Event Graphs. *Communications of the ACM,* 26: 957 - 963.

Stewart, J. (1999). Calculus (4th Edition). Pacific Grove: Brooks/Cole.

United States Navy. (1994). NAVEDTRA 14152, Mineman, Volume I. Retrieved November, 2003, from http://www.tpub.com/content/combat/index.htm.

Verrett, E. (August 1960). Effect of Shape and Depth on Wave Forced Oscillations of Submerged Moored Objects. M.S. Thesis, Massachusetts Institute of Technology, Cambridge, MA.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Arnold Buss
   Department of MOVES
   Naval Postgraduate School
   Monterey, California

4. Professor Susan Sanchez
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

5. LCDR Russell Gottfried
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

6. Doug Horner
   Department of Mechanical Engineering
   Naval Postgraduate School
   Monterey, California

7. CAPT Kenneth Voorhees
   Comptroller
   Puget Sound Naval Shipyard
   Bremerton, Washington